

UNIVERSITY OF CALGARY

Segmentation-Based Repair of Unstable Procedurally Generated Game Levels in Angry Birds

by

Mahdi Farrokhmaleki

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA

APRIL, 2026

© Mahdi Farrokhmaleki 2026

Abstract

The video game industry has been expanding rapidly, and this huge market means there is always a need for new content. However, the process of creating a game is very time-consuming and can take several years. Procedural Content Generation (PCG) is considered one of the solutions to this problem and can increase replay value, reduce production costs, and minimize effort. While PCG methods can generate vast amounts of content efficiently, ensuring structural stability remains a critical challenge, particularly in physics-based games such as *Angry Birds*. In these games, levels consist of interconnected structures, where even minor instability can lead to unintended collapses. This thesis presents a segmentation-driven repair framework designed to enhance the stability and structural quality of AI-generated game levels for industry use.

The proposed framework operates in two main stages: (1) detection of structural gaps using semantic segmentation, and (2) automated reinforcement through targeted block placement. Three segmentation architectures (U-Net, SegFormer, and YOLOv8m-Seg) were trained and evaluated on a dataset of unstable *Angry Birds*-style levels. Performance was measured using Precision, Recall, F1-score, and Intersection over Union (IoU) for segmentation accuracy, as well as Block Velocity (BV), Block Damage (BD), and Block Destruction (BDes) metrics for stability improvement. Manual evaluation conducted by the author indicated that YOLOv8m-Seg repairs produced the most stable and structurally consistent results to assess perceived stability and structural coherence.

Experimental results show that all three models contributed to substantial stability gains,

with YOLOv8m-Seg achieving the highest overall performance. Specifically, YOLOv8m-Seg repairs increased the proportion of stable levels up to **45%** according to level stability metrics, while maintaining structural consistency. Manual evaluation conducted by the author indicated that YOLOv8m-Seg repairs produced the most stable and structurally consistent results, followed by SegFormer and U-Net.

The contributions of this thesis include the development of a modular repair framework for AI-generated levels, a comparative study of segmentation architectures for structural gap detection, and the demonstration of repair strategies that prioritize stability improvements while preserving the original structural design of levels. The proposed framework can be integrated into industry pipelines as a post-processing, designer-assist, or automated QA tool, paving the way for broader adoption of AI-driven content generation in professional game development.

Preface

This thesis is an original work by the author. Portions of this thesis are based on research previously published in peer-reviewed conference proceedings. These publications form the core technical contributions of the thesis and have been expanded, revised, and integrated into a unified narrative that addresses the overarching research objective of improving the structural stability of AI-generated game levels for industry applications. In this thesis, we focus on structural stability as the primary objective. Throughout this work, the term high quality refers specifically to levels that are structurally stable and physically consistent under simulation.

The following papers are included:

1. **Farrokhmaleki, M., Rahmati, P., & Zhao, R.** (2025). From Unstable to Playable: Stabilizing Angry Birds Levels via Object Segmentation. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

This paper constitutes the primary technical contribution of the thesis, forming the basis for Chapters 3, 4, and 5. It introduces the complete segmentation-driven repair framework, presents a comparative analysis of multiple deep learning architectures, and provides a thorough evaluation of the pipeline’s effectiveness. The work is expanded within the thesis to include a more detailed methodological discussion and broader implications for the games industry. The author (Farrokhmaleki) was the lead researcher and was responsible for conceptualizing the research problem, developing the repair framework, implementing the models, conducting the experiments, analyzing the results,

and writing the majority of the manuscript. The second author (Rahmati) assisted with the manual evaluation of repaired levels and contributed to writing portions of the background section.

2. **Farrokhmaleki, M., & Zhao, R.** (2024). Procedural Content Generation in Games: A Survey with Insights on Emerging LLM Integration. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 20, 167-178.

This survey paper provides the foundational context and motivation for the thesis. Its analysis of the current state of Procedural Content Generation identifies the research gap in automated content repair that this thesis addresses. The findings from this paper directly inform the Introduction (Chapter 1) and the Literature Review (Chapter 2).

In both cases, the author was the primary contributor, responsible for conceptualizing the research problem, designing the methodology, implementing the experiments, analyzing the results, and writing the majority of the manuscript.

The integration of these works into the thesis involved harmonizing notation, expanding literature review sections, adding unpublished experimental results, and developing a more comprehensive discussion of the implications and limitations of the approach.

Image Credits

All illustrations, diagrams, and data visualizations in this thesis were created by the author unless otherwise indicated below. Graphs were generated using `Matplotlib`, and diagrams were created using `draw.io`. Figures reproduced or adapted from prior work or external sources are explicitly identified and attributed.

- Figures 2.2, 2.3, 3.2, 3.3, 4.1, 3.4, 1.1a, and 1.1b are either original figures created for the author’s publication or figures previously published and cited in that paper, and are reproduced here with appropriate attribution.
- The in-game images in Figure 1.2 and 3.1 are screenshots from the *Science Birds* framework, a research platform based on the commercial game *Angry Birds*, and are used for academic and illustrative purposes.
- The image in Figure 2.1a is adapted from Solkoll (Wikimedia Commons) and is used under a Creative Commons license.
- The image in Figure 2.1b is adapted from *NetHack* and is used under the NetHack General Public License.
- The image in Figure 2.4 is adapted from Newe and Ganslandt (2013), *PLoS ONE* 8(11): e79004, and is used under the Creative Commons Attribution (CC BY) license.
- The image in Figure 2.5 is adapted from M. Theiler’s original photograph *Schreibtisch-mit-Objekten.jpg* and shows objects detected using OpenCV’s Deep Neural Network

(DNN) module with a YOLOv3 model trained on the COCO dataset. The image is the author's own work and is used with attribution.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Dr. Richard Zhao, for his invaluable guidance, encouragement, and expertise throughout the course of this research. His constructive feedback and unwavering support have been instrumental in shaping both the technical direction and overall quality of this work. I am also grateful to my co-author, Parsa Rahmati, on the AIIDE 2025 paper, whose contributions provided essential foundations for this thesis. I would like to sincerely thank my committee members, Dr. Farhad Maleki, Dr. Frank Maurer, and Dr. Faramarz Samavati, for their time, support, and valuable feedback.

This research would not have been possible without the dataset and tools that supported the experiments. I also acknowledge the game development and AI research communities for fostering open collaboration and innovation.

I would also like to acknowledge the people of Iran, my home country, whose resilience and pursuit of a better future continue to inspire me.

Finally, I am deeply thankful to my family, my partner, my friends, and my dear nephew Araz for their unwavering support, encouragement, and patience throughout all these years. Their belief in me has been a constant source of motivation during my studies and research.

Table of Contents

Abstract	i
Preface	iii
Image Credits	v
Acknowledgments	vii
Table of Contents	x
List of Tables	xi
List of Figures and Illustrations	xiii
1 Introduction	1
1.1 Thesis Motivation	1
1.2 Thesis Objective	4
1.3 Research Questions	5
1.4 Thesis Contributions	5
1.5 Thesis Organization	6
2 Literature Review	8
2.1 Procedural Content Generation (PCG)	8
2.2 Content Types	9
2.3 Physics-Based Game Level Generation	11
2.3.1 Characteristics of Physics-Based Levels	12
2.3.2 Existing Physics-Based Generation Approaches	12
2.4 Angry Birds Level Generation	12
2.5 PCG Algorithms	15
2.5.1 Search-Based Methods	15
2.5.2 Machine Learning-Based Methods	17
2.5.3 Other Methods	21
2.5.4 Large Language Models	23
2.5.5 Combined Methods	25
2.6 PCG Gaps and Limitations	25
2.6.1 PCG Challenges in Physics-Based Games	26

2.7	Level Repair in Procedural Content Generation	27
2.7.1	Manual Repair Approaches	27
2.7.2	Automated Repair Methods	28
2.8	Object Segmentation	28
2.8.1	U-Net	29
2.8.2	SegFormer	30
2.8.3	YOLOv8m-Seg	31
3	Methodology	33
3.1	Repairing Angry Birds Levels	33
3.2	Model Architecture Selection	34
3.2.1	Comparison of the Three Architectures	35
3.3	Dataset	35
3.4	Training Details	38
3.4.1	Training Configuration	38
3.5	Repair Process	39
3.5.1	Stage 1: Stability Evaluation	39
3.5.2	Stage 2: Automated Repair Operations	40
3.5.3	Stage 3: Re-evaluation of Repairs	41
3.5.4	Design Choices and Rationale	42
3.5.5	Illustrative Example	42
4	Evaluation and Results	44
4.1	Evaluation Metrics	44
4.1.1	Model Performance Metrics	45
4.1.2	Level Stability Metrics	47
4.2	Comparing Model Architectures	48
4.2.1	Interpretation of Trade-offs	50
4.2.2	Implications for the Repair Pipeline	50
4.3	Efficacy of the Repair Pipeline	51
4.3.1	Quantitative Repair Success	51
4.3.2	Analysis of Unsuccessful Repairs	52
5	Discussion	54
5.1	Interpretation of Findings	54
5.1.1	Impact of Repairs on Structural Properties	55
5.1.2	Industry Applicability	56
5.2	Relation to Research Objectives	57
5.3	Limitations	59
6	Conclusion	62
6.1	Summary of Work	62
6.2	Research Questions Revisited	62
6.3	Key Contributions	64
6.4	Implications for Industry	65

6.5	Future Work	66
6.6	Closing Remarks	67

List of Tables

2.1	Block types that are available in Science Birds.	13
4.1	Segmentation model performance	49
4.2	Summary of Repair Outcomes	52
5.1	Objectives–results mapping	60

List of Figures

1.1	The evolution of procedural level generation. (a) An early example using simple, rule-based algorithms and ASCII characters. (b) A modern example generated by a machine learning model that learns complex patterns from data.	2
1.2	An example of structural instability in an AI-generated level for <i>Angry Birds</i> game. This unstable level collapses upon loading due to critical gaps in its support structure.	3
2.1	Examples of procedurally generated content.	9
2.2	A timeline showing the types of algorithms that appeared in PCG-related research papers from 2019 to 2023.	11
2.3	A timeline breaking down research published 2019-2023 in select game-related conferences on the topic of PCG, sorted by targeted content type. A few noteworthy LLM-based works are pointed out.	24
2.4	Segmented 3D model of a left human femur generated using MeVisLab, showing the outer surface (red), the interface between compact and spongy bone (green), and the bone marrow surface (blue).	29
2.5	Objects detected using OpenCV’s Deep Neural Network (DNN) module with a YOLOv3 model trained on the COCO dataset. The model detects 80 common object categories.	31
3.1	Screenshot of a level from the <i>Science Birds</i> framework, a research platform based on the gameplay rules and physics mechanics of the commercial game <i>Angry Birds</i> .	34
3.2	The automated level repair pipeline. The top diagram (a) illustrates the high-level workflow, where an AI-generated level is first simulated and evaluated for stability. If the level is deemed unstable, it enters the repair stage. After the repair is applied, the level is re-evaluated to confirm its stability. The bottom diagram (b) provides a detailed breakdown of the “Repair Sub-Process” block, showing the five steps from encoding the level into an image, using the model to predict gaps, filling the gaps with wood, and decoding the result back into a new XML file.	36
3.3	A visual demonstration of the YOLOv8m-Seg model’s gap detection capability. (a) The binary image of an unstable level is given as input. (b) The model outputs a segmentation mask (shown as red overlays) that identifies the location and shape of structural gaps.	37

3.4	Illustration of a successful level repair: (a) The AI-generated level is flagged as unstable. (b) The physics simulation confirms instability as the structure collapses. (c) After our system repairs the level, it becomes stable.	43
4.1	Training Progress of YOLOv8m-Seg (blue), SegFormer (orange), and U-Net (green) models over epochs.	46
4.2	Comparison of mask-based mean Average Precision (mAP) across YOLOv8 variants.	51

Chapter 1

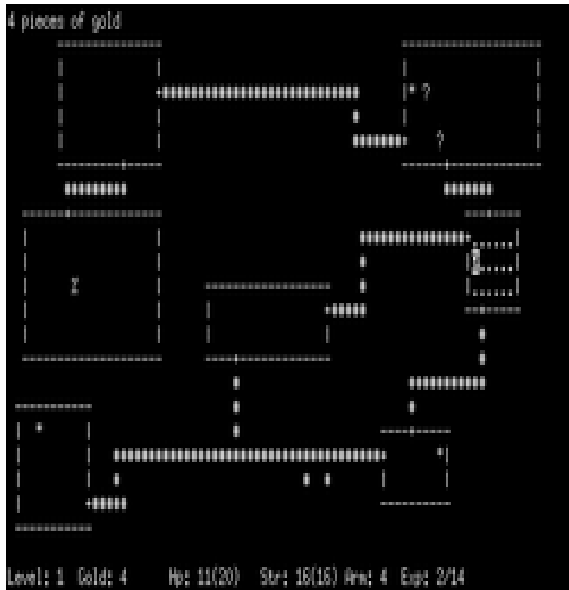
Introduction

1.1 Thesis Motivation

Procedural Content Generation (PCG) is the algorithmic creation of game content, such as **game space** (levels, maps), **game bits** (textures, assets), and **game scenarios** (narratives, quests), often with limited or indirect user input [1, 2]. In the modern video game market, which is one of the largest entertainment industries worldwide, the demand for new, high-quality content is relentless. As the manual creation of game worlds can be incredibly time-consuming and costly, PCG plays a critical role in reducing development costs, increasing replay value, and enabling the creation of diverse environments at scale [3].

Historically, PCG was implemented through handcrafted rules and generative grammars, as seen in early titles like *Rogue* (1980). The field has since evolved to incorporate more sophisticated techniques, including **search-based algorithms** like evolutionary computation, and more recently, advanced machine learning methods [4].

Recent advances in deep learning have significantly expanded the capabilities of PCG. Models such as Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and Large Language Models (LLMs) can now produce a wide array of game content by learning from large datasets of human-authored examples [5, 6]. Despite these advancements,



(a) A generated dungeon from *Rogue* (1980).



(b) A generated level for *Super Mario Bros.*

Figure 1.1: The evolution of procedural level generation. (a) An early example using simple, rule-based algorithms and ASCII characters. (b) A modern example generated by a machine learning model that learns complex patterns from data.

a persistent challenge remains: ensuring that generated content meets industry-quality standards. Many powerful generative models produce content with functional or aesthetic flaws, creating a gap between raw AI output and industry-ready assets.

This quality-assurance challenge is especially pronounced in physics-based games, such as *Angry Birds*, where levels consist of interconnected structures that must be physically stable to remain structurally reliable. Even minor instabilities can lead to unintended collapses when a level is loaded, rendering the content useless. While models like GANs can produce visually convincing structures, they typically lack explicit stability constraints, leading to a low yield of usable levels due to flaws like floating blocks and misaligned elements [5]. This inefficiency forces designers into a cycle of generating, testing, and discarding large volumes of content, which requires extensive manual curation and undermines trust in automated tools.

Although PCG has advanced considerably, most research has focused on *generating* new content rather than *improving* it. While some automated repair techniques exist, they



Figure 1.2: An example of structural instability in an AI-generated level for *Angry Birds* game. This unstable level collapses upon loading due to critical gaps in its support structure.

have largely focused on simpler, tile-based games using rule-based correction. The repair of complex, physics-based structures remains a significant and underexplored challenge. The existing body of research, including recent surveys of the field, reveals several critical gaps:

- **Lack of repair-focused frameworks:** Few studies address the automated repair of unstable AI-generated levels, especially in the context of physics-based games.
- **Limited use of computer vision for stability analysis:** The application of segmentation for identifying structural flaws in game levels remains underexplored.
- **A disconnect from industry needs:** There is a notable gap between academic work on generation and the practical needs of the games industry, which requires robust, high-quality content.

This thesis is motivated by the need to bridge the gap between raw AI-generated content and industry-ready game levels. It directly addresses the lack of automated repair tools—a key direction for future work identified in surveys of the field—by introducing a segmentation-based framework that automatically detects and corrects structural instabilities. By salvaging otherwise discarded content, this work aims to increase the yield of structurally stable levels

from modern PCG systems. In this thesis, high quality refers specifically to structural stability and physical consistency under simulation.

1.2 Thesis Objective

The overarching goal of this thesis is to enhance the industry-readiness of AI-generated game levels by improving their structural stability through automated repair. This is achieved by designing, implementing, and evaluating a segmentation-based repair pipeline capable of detecting and correcting structural instabilities in levels generated by machine learning models.

The specific objectives of the research are as follows:

1. **Survey the landscape of Procedural Content Generation (PCG) techniques**, including traditional algorithms, deep learning methods, and emerging Large Language Model (LLM)-based approaches, to identify gaps in repair-focused research.
2. **Design and develop a segmentation-based repair framework** capable of identifying and localizing destabilizing structural gaps in physics-based game levels.
3. **Compare multiple segmentation architectures** (U-Net, SegFormer, YOLOv8m-Seg) to identify the most effective model for gap detection in unstable levels.
4. **Implement automated repair strategies** to fill detected gaps with appropriate structural elements, ensuring improved stability without significantly altering the original design.
5. **Evaluate the effectiveness of the repair pipeline** using multiple stability metrics (*Block Velocity*, *Block Damage*, and *Block Destruction*) and quantify improvements in level stability.
6. **Explore the potential for generalization** of the repair approach to other game genres and instability types.

1.3 Research Questions

This thesis is driven by the need to address the quality limitations of AI-generated game content. In particular, it investigates the following central and supporting research questions:

- **Main Question:** What are the primary problems and limitations in AI-generated game content, particularly regarding the structural stability of levels in physics-based games?
- **RQ1:** Can we automatically repair unstable AI-generated game levels to improve their stability without extensive human intervention?
- **RQ2:** How can segmentation-based computer vision techniques be leveraged to reliably detect structural instabilities in physics-based game levels?
- **RQ3:** Which deep learning segmentation architectures (e.g., U-Net, SegFormer, YOLOv8m-Seg) are most effective for detecting destabilizing gaps and flaws in generated levels?
- **RQ4:** To what extent can an automated repair pipeline improve the yield of stable levels from modern Procedural Content Generation systems, and can this approach generalize to other game genres and instability types?

1.4 Thesis Contributions

This thesis makes the following primary contributions to the field of Procedural Content Generation and automated game content curation:

1. **A comprehensive survey of modern PCG techniques**, which formally identifies automated repair as a critical and underexplored area of research. This contribution establishes the foundational context and motivation for the work.

2. **The design and development of a novel, segmentation-driven framework** for automatically detecting and repairing structural instabilities in AI-generated, physics-based game levels.
3. **A comparative analysis of multiple deep learning architectures** (U-Net, SegFormer, and YOLOv8m-Seg) for the specific task of structural flaw detection, providing insight into the most effective models for this application.
4. **Empirical validation of the repair pipeline**, demonstrating that it can increase the yield of structurally stable content from a standard GAN-based generator by up to 45% under established stability metrics, as verified through simulation and manual evaluation conducted by the author.

The source code and supplementary materials for this project are publicly available on GitHub at

<https://github.com/mahdifarro/space-finder>.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 reviews the current state of research in Procedural Content Generation (PCG), including machine learning approaches and prior work in content repair, and surveys the metrics used to evaluate stability in physics-based games. Following this, Chapter 3 presents the proposed segmentation-based repair framework in detail, including the dataset preparation, the segmentation architectures evaluated, and the automated repair strategies. Chapter 4 then describes the experimental setup and presents the quantitative results from the repair pipeline, including a detailed performance comparison of the segmentation models and an analysis of stability improvements. In Chapter 5, these findings are interpreted, and their implications for both research and the games industry are discussed, along with the limitations of the current work. Finally,

Chapter 6 summarizes the key contributions of the thesis and outlines promising directions for future research in the automated repair of AI-generated content.

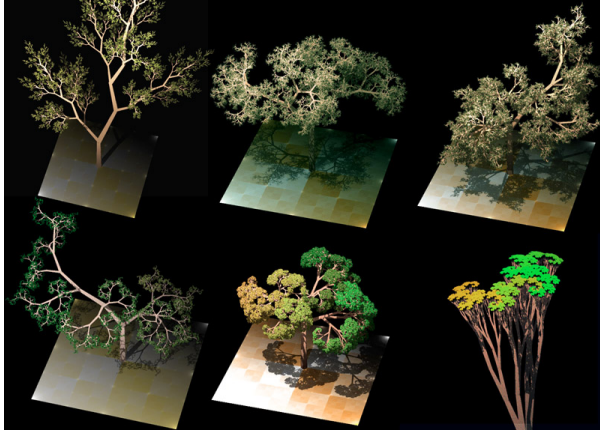
Chapter 2

Literature Review

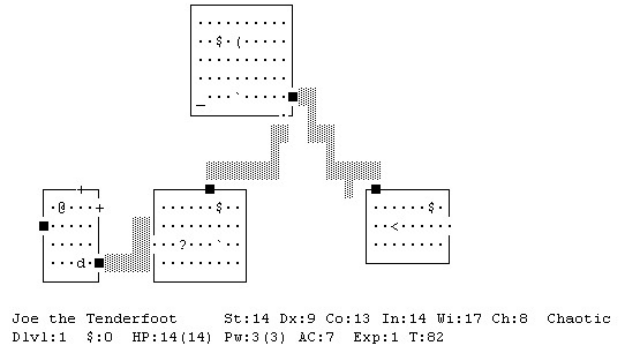
This chapter reviews research relevant to the automated repair of AI-generated game levels, with an emphasis on physics-based games. It first summarizes procedural content generation (PCG) methods and the types of content they target. It then focuses on physics-based level generation, with Angry Birds as the primary case study. Next, it surveys limitations of existing PCG approaches, highlighting the structural stability issues that motivate post-generation repair. Finally, it reviews prior work on level repair and stability evaluation, establishing the technical foundation for the segmentation-based repair framework proposed in Chapter 3.

2.1 Procedural Content Generation (PCG)

The video game industry has been expanding rapidly and even surpassed the combined revenue of the music and movie industries in 2022 [7]. This market growth increases demand for high-volume, high-quality content. However, building commercial game content remains resource-intensive and time-consuming, and can take several years [8]. Procedural Content Generation (PCG) is considered one of the solutions to this problem and can increase replay value, reduce production costs, and reduce manual authoring effort [9]. PCG for games has existed since the 1980s, and it was mainly used in roguelike games such as *Beneath Apple*



(a) Fractal trees generated using an L-system, including a varied dragon curve and a Heighway dragon curve.



(b) A procedurally generated dungeon map from the video game *NetHack*.

Figure 2.1: Examples of procedurally generated content.

Manor (1978) and the genre’s namesake, *Rogue* [10].

PCG can be used to create a variety of content, but it is commonly used to create art assets [11, 12], maps and levels [13, 14], game mechanics [15], and music for games [16]. The algorithms used can greatly vary depending on the content they are supposed to generate, but we can generally categorize them under a few categories. Most of the algorithms reviewed here fall under one or a combination of three categories: (1) search-based methods, such as Monte Carlo Tree Search (MCTS), which focus mainly on optimizations, (2) learning-based methods, including traditional machine learning and deep learning (DL), such as generative adversarial networks (GAN) and reinforcement learning (RL), which are the recent additions to PCG, and (3) other methods, such as noise functions and generative grammars, that we could not categorize in the earlier categories, because of the massive interest in DL in recent years, there are many papers published that use DL algorithms as part of PCG.

2.2 Content Types

Almost everything, from sounds to the game narrative, can be generated nowadays, but the generated content for games can vary a lot depending on the algorithms used. For example,

LLMs are mostly used to create game narratives [17], and GANs are considered a better solution to generate images [11] and 2D levels [18].

There are several ways that we can categorize generated content. For example, it can be divided into online and offline generation. Online generation means that the content generation is performed during the runtime of the game, while offline generation means it is created during game development. Additionally, content can be categorized as necessary or optional [19]. For this survey, we use categories similar to the ones presented by Hendrikx et al. [20] with some modifications. We divide content created for games into five different categories, each consisting of multiple items.

1. **Game bits:** This category consists of the smallest pieces (units) used in games. Any type of generated texture, sound, vegetation, structures and buildings, and object properties (e.g. can it interact with basic physics?) goes into this list. We also included art and images [21] created by PCG algorithms in this category.
2. **Game Space:** Game space is the physical environment in which the game takes place and is created from game bits. PCG algorithms are commonly used to generate maps and roads for games. Researchers have even tried to create entire game worlds with them [22].
3. **Game Scenarios:** Game scenarios are parts of the game that are tied to the narrative. This includes generated conversations, stories, and quests. We include puzzles and interactive level elements in game scenarios because they are part of the scenarios that contribute to the story. This is especially true in the case of generated levels for side-scrolling 2D games [14]. Levels for music games, such as Guitar Hero or Dance Dance Revolution, can be seen as 2D levels as well [23].
4. **Game Design:** This category consists of any mechanics or rules created for games (what can the player do, and what are the goals?) It also includes a generated system design, which we interpret as systems used in the games. Generating spawn points

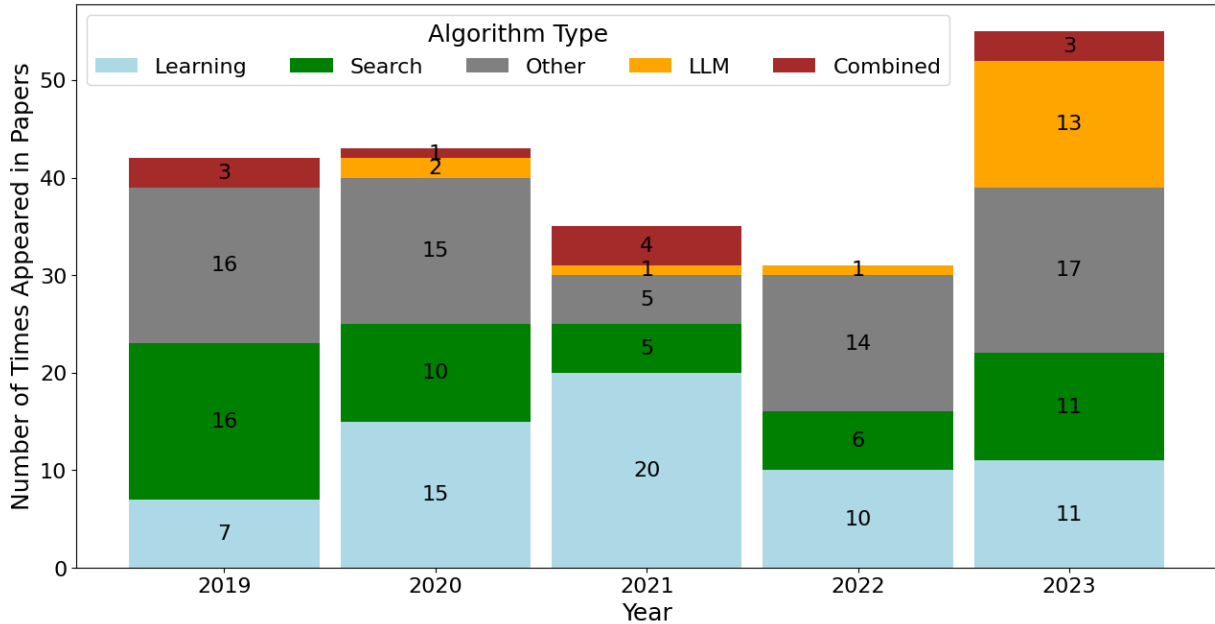


Figure 2.2: A timeline showing the types of algorithms that appeared in PCG-related research papers from 2019 to 2023.

for first-person shooter games for game design purposes [24] is a perfect example of a generated design system.

5. **Derived content:** Derived content includes everything that is not essential to the game but can help the player better immerse in the game world. This category contains background NPC interactions, news found within the game, and chatter of different characters that are not part of the game’s story or a quest.

2.3 Physics-Based Game Level Generation

Physics-based games present unique challenges for Procedural Content Generation (PCG) due to their reliance on realistic simulation of object interactions, material properties, and structural stability. Unlike tile-based or purely visual content generation, where aesthetic or layout rules may suffice, physics-based levels must adhere to physical constraints to ensure structural stability.

2.3.1 Characteristics of Physics-Based Levels

Physics-based levels often consist of composite structures made from elements with different material properties (e.g., wood, stone, ice in *Angry Birds*). The position, orientation, and interconnection of these elements directly influence the stability of the level. Even minor deviations—such as slight misalignments, unsupported overhangs, or unfilled gaps—can result in unintended collapses when the level is loaded into the game engine.

In industry settings, stability is critical not only for structural reliability but also for the perceived quality of generated levels. A level that collapses before player interaction may be perceived as broken, undermining the perceived polish and reliability of the game.

2.3.2 Existing Physics-Based Generation Approaches

Early work on physics-based PCG relied on handcrafted templates or parametric generators that ensured stability by design. While effective, these approaches limited diversity and novelty. More recent efforts employ machine learning models, such as Generative Adversarial Networks (GANs), to learn design patterns from large datasets of human-created levels. However, these models typically lack embedded physics knowledge, leading to frequent generation of unstable structures.

For example, GAN-generated *Angry Birds* levels often contain floating blocks, insufficient support for load-bearing elements, and gaps in critical structural areas. This results in a high proportion of generated content being unsuitable for direct use in production pipelines without additional manual curation or automated repair.

2.4 Angry Birds Level Generation

Angry Birds is a 2D physics-based game where players launch birds at block-based structures to eliminate pigs. The game mechanics rely on destructible environments, where structures are usually composed of wood, ice, or stone blocks, each having different resistance and

attributes. Players must strategically aim their limited number of birds to cause structural collapse and maximize damage to eliminate all pigs.

Because the original *Angry Birds* game is not open source, most academic research and AI-driven level generation studies utilize a Unity-based clone called Science Birds [25]. Science Birds replicates the mechanics of the original game while allowing researchers to generate, modify, and analyze levels programmatically. Each level in Science Birds is stored in an XML format, describing the position, type, and orientation of every game object.









ID	Shape	Name	Dimensions
1		SquareHole	(0.85, 0.85)
2		RectBig	(2.06, 0.22)
3		RectMedium	(1.68, 0.22)
4		RectSmall	(0.85, 0.2)
5		RectFat	(0.85, 0.43)
6		RectTiny	(0.42, 0.22)
7		SquareTiny	(0.22, 0.22)
8		SquareSmall	(0.43, 0.43)

Table 2.1: Block types that are available in Science Birds.

Table 2.1 presents the standard block types used in *Angry Birds* levels, categorized by shape and material. While the game also includes irregular block types, such as triangular and

circular blocks, these are typically excluded from automated level generation due to complex stability constraints. Most PCG-based approaches for *Angry Birds* generate structures using only rectangular blocks with limited block orientations (0° or 90° angles) to minimize instability.

Over the past decade, researchers have explored multiple approaches for automated level generation in *Angry Birds*. Many of these methods have been showcased in the AIBirds Level Generation Competition [26]. The most common techniques include:

- Genetic Algorithms (GA): Uses evolutionary principles to iteratively improve level design [25].
- Search-Based Approaches: Generates levels by optimizing specific structural constraints [27, 28].
- Monte Carlo Tree Search (MCTS): Simulates multiple playthroughs to identify optimal level configurations [29].
- Variational Autoencoders (VAE): Uses deep learning to generate levels from latent representations [30].

Some studies have focused on specific objectives, such as:

- Generating levels that resemble text, quotes, or formulas [31].
- Creating deceptive structures that mislead players [32].
- Designing Rube Goldberg-style contraptions for dynamic interactions [33].

Despite these advancements, ensuring stability remains a challenge, as AI-generated levels can often contain floating blocks, structurally unsound elements, or unintended difficulty spikes. This limitation highlights the need for post-processing techniques (such as the repair approach proposed in our study) to refine AI-generated levels and improve their structural stability.

2.5 PCG Algorithms

Due to the different types and roles of content in games, diverse PCG methods have been adapted for procedural content generation. In this section, we present different algorithms that exist and can be used to generate items. We categorized them into five main categories: Search-based, Machine Learning-based, Other, LLMs, and Combined Methods. The category called Combined Methods includes papers that use several methods in their study, either in parallel or in one integrated system.

It is worth mentioning that there were a few methods that we could not place under specific subgroups, so we decided to mention them at the start of each category.

2.5.1 Search-Based Methods

The term ‘search-based PCG’ was coined by Togelius in his paper on the taxonomy of PCG [34]. A search-based PCG refers to a special case of generate-and-test PCG. A generate-and-test PCG does not directly dish out content it generates but instead tests the content first using a test function. Depending on the test result, the PCG can either accept the content or reject it and create new content.

A search-based PCG is a test-and-generate PCG that satisfies two criteria [35]. First, instead of simply accepting or rejecting content, the test function of a search-based PCG assigns a real value that measures the acceptability of the content. This value is often called fitness, and the function that produces it is called a fitness function. Second, in the creation of new, better content, a search-based PCG uses the previously rejected content as the creation base, and the new content is a slightly modified version of the old content. Search-based methods have been used to generate a variety of content, such as puzzles, race tracks, levels, terrains, and maps [34, 35].

In many cases, search-based algorithms use some form of evolutionary algorithm as the main search mechanism, as evolutionary computation has so far been the method of choice

among search-based PCG practitioners. However, search-based PCG does not need to be married to evolutionary computation; other heuristic and stochastic search mechanisms are viable as well [34]. In this section, we discuss evolutionary algorithms, Wavefunction Collapse (WFC), Monte Carlo Tree Search (MCTS), simulated annealing, and particle swarm optimization.

1. **Evolutionary Algorithm:** In an evolutionary algorithm, a population of candidate content instances is held in memory. Each generation, these candidates are evaluated by the evaluation function and ranked. The worst candidates are discarded and replaced with copies of the good candidates, except that the copies have been randomly modified (i.e., mutated) and/or recombined [34]. One of the most famous examples of evolutionary algorithms is genetic algorithms. Genetic algorithms are used in many instances [36, 37, 38] to create playable levels and game bits. This algorithm can also be used to create more traditional types of games. For example, Botea and Bulitko [39] use this algorithm to create Romanian crossword puzzles. Other evolutionary algorithms are also used to create soundtracks for games [16], maps and game bits [16], roads [40], and levels for board games [41].
2. **Planning Algorithms:** Planning, in general, is a problem-solving technique consisting of a planning problem (i.e., initial state and goal specification) and a planning domain (i.e., objects, predicates, and action operators). Given the input of a planning problem, a sound planner produces a solution or a plan, which is a sequence of actions that achieve all the specified goal conditions without any causal threats [42]. In recent years, planning-based algorithms have often been used to generate stories in games because planning-based narrative generation is effective at producing stories with a logically sound flow of events [43].
3. **Wave Function Collapse (WFC):** WFC is a texture synthesis algorithm. Compared to earlier texture synthesis algorithms, WFC guarantees that the output contains only

those NxN patterns that are present in the input. This makes WFC perfect for level generation in games and pixel art, and less suited for large full-color textures [44]. The WFC algorithm also supports constraints, allowing it to be easily combined with other generative algorithms or manual creation [45].

4. **Monte Carlo Tree Search (MCTS):** Monte Carlo Tree Search (MCTS) is a heuristic search algorithm that involves searching combinatorial spaces represented by trees. In such trees, nodes denote states whereas edges denote transitions (actions) from one state to another [46].
5. **Simulated Annealing:** Simulated annealing is a process where a setup is randomly tweaked and compared to the previous setup using the cost function. If it is better, the new setup is kept. Otherwise, the decision to keep the new setup is made randomly, with the probability of keeping the old setup proportional to how much better it is [47].
6. **Particle Swarm Optimization (PSO):** PSO is a general-purpose optimization technique developed by Eberhart and Kennedy [48]. This technique was inspired by the concept of swarms in nature, such as bird flocking, fish schooling, or insect swarming. The idea is that individual members of the swarm can profit from the discoveries and previous experiences of all other members of the swarm during the search for the optimum solution [49].

2.5.2 Machine Learning-Based Methods

Machine learning methods have gained a lot of popularity during the last decade [50]. Research on neural networks under the name deep learning has precipitated a massive increase in the capabilities and application of methods for learning models from big data [51, 52]. They have provided us with new ways for generating audio, images, 3D objects, network layouts, and other content types [53] across a range of domains, including games. For example, long short-term memory (LSTMs) are mostly used for time-dependent sequential data (e.g.,

action sequences, agent paths, charts for rhythm) and language models [54], while generative adversarial networks (GANs) have been applied to generate artifacts, such as images, music, and speech [53]. Outside of content generation, machine learning algorithms have been widely used in testing generated maps using other algorithms [23]. Many other machine learning methods can also be utilized in a generative role, including n-grams, Markov models, autoencoders, and others [55, 56, 57, 50].

In this section, we talk about different machine learning methods such as Autoencoders (including deep variational autoencoders), Recurrent Neural Networks (including LSTMs), Generative Adversarial Networks (GANs), Markov Models, Reinforcement Learning methods, and Transformers.

The most recent entry, LLMs, are gaining a lot of attention among researchers. While state-of-the-art LLMs use transformers as their underlying architecture, their model-as-a-service (MaaS) nature puts them into a different category.

It is worth mentioning that not all articles regarding deep learning methods use complex neural network models. Merino et al. [58] use neural networks to create sprites and game maps for 2D games. Bhaumik et al. [59] and Kumaran et al. [14] use neural network models to generate maps and levels for 2D games.

1. **Simple Neural Networks:** Some works on PCG algorithms rely on the use of relatively simple neural networks. For example, papers such as Chen et al. [60] take images as inputs and output game levels using neural networks.
2. **Recurrent Neural Networks & LSTMs:** A recurrent neural network (RNN) is a type of artificial neural network that uses sequential data or time series data. They are distinguished by their “memory” as they take information from prior inputs to influence the current input and output [61]. LSTMs are similar neural networks introduced to eliminate the vanishing gradient problem in RNNs. LSTMs work to solve that problem by introducing additional nodes that act as a memory mechanism, telling the network when to remember and when to forget. As mentioned before, LSTMs and RNNs are

mostly used to create sequential data. Summerville and Mateas [62] deal with Mario levels as a sequence of data and combine Markov Chains and LSTMs to create new levels.

3. **Generative Adversarial Networks (GANs):** GANs are very popular for content generation purposes. They usually consist of two networks, a generator and a discriminator, that are trained iteratively to allow the generator to create more realistic content while the discriminator gets better at distinguishing generated content from real data [53, 23].

GANs are perfect for generating content represented by pixel-based images or 2D arrays of tiles, such as levels, maps, landscapes, and sprites. By reviewing the gathered papers, it can be seen that in the work using GANs for level generation, game levels are tackled as images only during training, while the constraints for validating levels are not considered at all.

4. **Markov Models:** The Markov chain algorithm is a typical constructive method. In this approach, content is generated on-the-fly [63] according to the conditional probabilities of the next state in a sequence based on the current state. This state can incorporate multiple previous states via the construction of n-grams. An n-gram model (or n-gram for short) is simply an n-dimensional array where the probability of each state is determined by the n states that precede it [50]. This is the simplest form of Markov chains, also called 1D Markov chains.

There are also multidimensional Markov chains (MdMCs) [64], where the state represents a surrounding neighborhood and not just a single linear dimension. A multidimensional Markov chain differs from a standard Markov chain in that it allows for dependencies in multiple directions and from multiple states, whereas a standard Markov chain only allows for dependence on the previous state alone [50]. In addition to their standard MdMC approach, Snodgrass and Ontañón developed a Markov random field (MRF)

approach [53] that performed better than the standard MdMC model in Kid Icarus, a domain where platform placement is pivotal to playability [65]. This method has generated novel but recognizable game names [66], natural language conversations, poetry, jazz improvisation [67], and content in a variety of other creative domains [68].

5. **Autoencoders:** An autoencoder is a type of neural network architecture designed to efficiently compress (encode) input data down to its essential features, then reconstruct (decode) the original input from this compressed representation. One of the famous autoencoders used for PCG is the variational autoencoder (VAE). VAEs are generative models that learn compressed representations of their training data as probability distributions, which are used to generate new sample data by creating variations of those learned representations [69]. These algorithms are mostly used for generating 2D maps and levels.

Snodgrass and Sarkar [70] use VAEs to generate level structures and a search-based approach to blend details from various platformers, while Sarkar et al. [71] directly train VAEs on levels from several platforming games and interpolate the latent vectors between domains for blending [23]. Sometimes, trained autoencoders may be used to repair unplayable levels. Davoodi et al. [72] train an autoencoder to repair manually designed levels for different games by re-iterating it over the decoder while using a trained discriminator from a GAN model to determine the stopping criteria [23].

6. **Reinforcement Learning:** RL problems involve learning how to map situations to actions that maximize a numerical reward signal [73]. Most articles that use RL develop agents to play generated levels, which indirectly serve as content evaluators.

To generate content using RL, the generation task is usually transformed into a Markov decision process (MDP), where a model is trained to iteratively select actions that would maximize expected future content quality. This transformation is not an easy task, and there is no standard way of handling it. Most RL PCG approaches require

an adaptation of the input to be used during generation [23]. There are also some interesting cases where RL can be used in the absence of data, provided a system for the learning agent to interact with can be set up [74].

7. **Transformers:** Transformers are a type of neural network architecture that transform an input sequence into an output sequence by learning context and tracking relationships between sequence components [75]. They rely on a self-attention mechanism [76], which facilitates the capture of intricate semantic relationships within high-dimensional feature spaces. For example, PCGPT [77] used transformers to iteratively generate complex and diverse game maps in the Sokoban game.

2.5.3 Other Methods

There are some methods that do not belong in the previous categories. We added this list for frequently used methods in PCG that we could not fit in any other category. It includes pseudo-random number generators (PRNGs), generative grammars, generative graphs, and fractals.

It is worth mentioning that these categories are not the only means of generating content for games. There are some articles about innovative ways of PCG. For example, Wootton [78] uses quantum blur effects to create maps and levels for various games. Wootton [79] also uses quantum computation combined with graphs to generate maps.

1. **Pseudo-random Number Generator (PRNG):** One of the simplest and earliest approaches to procedural game content generation is based on pseudo-random number generation (PRNG). PRNG is an algorithm for generating a sequence of numbers that approximates the properties of random numbers [80]. A PRNG algorithm consists of three parts: the seed, which is the initial value, the formula, which converts the seed to output, and the distribution, which is the variance of the results [81].

PRNG was first used as a data compression method because the generated sequence,

while appearing random, can be reproduced if the same seed and algorithm are used. Combined with other methods, PRNG-based techniques can be used to generate buildings, textures, and items [20]. One of the most famous forms of PRNGs is noise functions. Perlin noise and other noise functions are commonly used for texture generation. Noise-generated textures can be mapped easily on complex objects, unlike raster 2D images. The implementation of noise is relatively simple and is present in many software shaders and hardware graphics cards, such as NVIDIA's [20]. One of the main drawbacks of this method is that images generated randomly pixel-by-pixel have no meaningful structure. Many procedural techniques address this issue by finding the balance between iterative generation and random generation.

2. **Generative Grammars:** Generative grammars, stemming from Noam Chomsky's study of languages in the 1960s [20], are sequences of words (or "sentences") with rules regarding how and when to replace some words with other words. They can be used to create correct objects from elements encoded as letters/words. L-systems, split grammars, wall grammars, and shape grammars, which have been used for plant generation, linear map dungeon or story generation, and quest generation, are all part of generative grammars. The downside of using this method is that it is linear, so it cannot be used to generate a non-linear story. The possible solution is using generative graphs, which we discuss in the following section.

3. **Generative Graphs:** A graph is a set of vertices connected by edges. Generative graphs are used as a solution for the linearity problem of generative grammars. One well-known type of generative grammar is graph grammars. A graph grammar is a set of rules that modify a graph. It is a framework introduced decades ago [82, 83].

One of the challenges of using graph grammars was the need for an expert to design the rules, but now, some researchers [84] have worked on creating the grammar automatically. The content generated using generative graphs is almost similar to that generated by

generative grammars, only they do not have to be linear. One of the most famous products of graph grammars is SpeedTree [83], a well-known set of tools to generate trees in the entertainment industry [85].

4. **Fractals:** The term "fractal" was coined and popularized by Benoit B. Mandelbrot [86]. It describes a broad set of shapes characterized by non-integer dimension or an interesting mismatch of dimension (Hausdorff dimension strictly exceeding topological dimension), and detail at all scales or self-similarity. Fractals are frequently used in procedural content generation because self-similarity seems to mimic natural processes such as erosion and plant growth. The subdivision method also maps well onto level of detail implementations, allowing an 'infinite' amount of detail to be included by recursively subdividing the detail shown as the viewpoint moves closer to the fractal object [87].

2.5.4 Large Language Models

In recent years, there has been an explosion of research on the applications of LLMs, and PCG is no exception. We found 17 papers from 2019 to 2023 using LLMs as part of their pipeline (with 3 other papers using combined methods that integrated LLMs). Referring back to Figure 2.2, we see that in 2023, the use of LLMs drastically increased compared to previous years (13 papers and 2 other papers with combined methods integrating LLMs). This coincides with the release of ChatGPT and its underlying model, GPT-3. These models are used to create narratives, NPC chatter, and even mechanics. A famous example is *Dungeon 2*, a text adventure game [88]. In this game, players can type in any command and the system can respond to it reasonably well, creating the first never-ending text adventure. The system is built on OpenAI's GPT-2 language model [89], which was further fine-tuned on a number of text adventure stories. *1001 Nights* is also another project that uses GPT as one of the main mechanics of its gameplay [90]. Language models are also used in role-playing board games as an assistant for the game masters [91]. LLMs can also be used to create game levels.

SCENECRAFT [92] is a framework that transforms high-level natural language instructions from authors into dynamic game scenes that include NPC interactions, dialogue, emotions, and gestures. The research by Todd et al. [93], Nasir and Togelius [94], and Sudhakaran et al. [95] also focus on using LLMs to create levels for games. For example, the latter uses MarioGPT, a fine-tuned GPT-2 model designed to generate Super Mario Bros levels based on textual prompts.

While current LLMs use transformers as their underlying model, with the publication of GPT-3, a new form of service has emerged. “Model as a service” (MaaS) involves deploying a model on a cloud-based infrastructure and offering its functionalities through APIs or web interfaces. These commercialized models, such as GPT-3, are often no longer open for researchers to explore and study their underlying architecture, creating a black box that is only known to a select few. Our review shows that research published on using GPT-3 and its successors is no longer about training a new model, but about best ways to use an existing pre-trained model that is restricted by a private entity in many ways. While open source LLMs do exist, in our review, the vast majority of LLM usage was still with GPT-based LLMs.

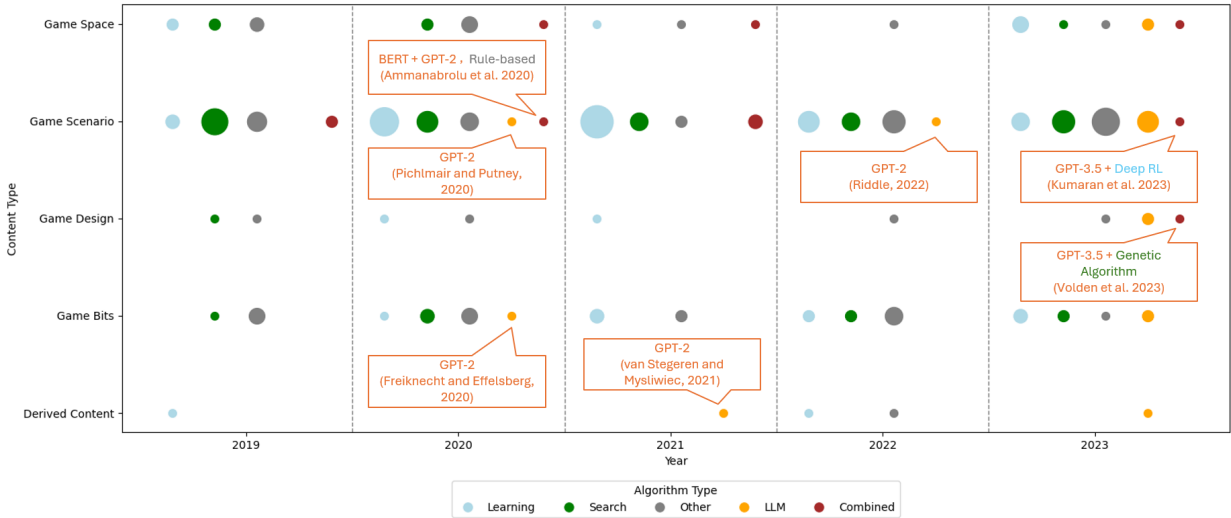


Figure 2.3: A timeline breaking down research published 2019-2023 in select game-related conferences on the topic of PCG, sorted by targeted content type. A few noteworthy LLM-based works are pointed out.

2.5.5 Combined Methods

So far, we have discussed four different methods used for generating content in games. However, these methods can be combined. Many researchers have worked on using evolutionary algorithms with machine learning methods. One of the most popular examples of combining methods is the Latent Variable approach [96], which combines unsupervised learning in the form of a GAN or VAE with evolutionary computation to search for content in the learned space of a GAN/VAE. In the context of games, this approach has been employed to generate 2D game levels like Super Mario Bros and Zelda levels [88, 89]. RL is also used in combination with many different algorithms. For instance, Kumaran et al. [97] creates game levels through a natural language interface and evaluates the levels using RL. Instead of relying solely on one method, combining different methods can be very effective in generating new content. RL models and search-based algorithms are effective tools to repair generated content (especially levels and structures) created by other algorithms (like GANs and LSTMs). More recently, combined methods integrating LLMs have also begun to emerge [97, 98]. While PCG research tends to focus on using existing methods in innovative ways, these combined methods form a rather emerging approach to solving complex problems.

2.6 PCG Gaps and Limitations

Despite the rapid growth of Procedural Content Generation (PCG) research, several gaps and limitations remain that hinder the field’s ability to fully meet industry needs.

A key limitation is the overwhelming focus on 2D content generation. As highlighted in our analysis, most existing works deal with 2D level generation, particularly in platformers such as Super Mario Bros.. While this line of research has produced a wide range of creative approaches, it falls short of addressing the complexity of 3D game environments, which dominate the commercial games industry. Generating stable and coherent 3D levels requires more advanced structural reasoning, spatial representation, and physics integration—areas

where research remains relatively sparse.

Another notable gap is the disconnect between academic contributions and industry applications. Many published papers introduce novel algorithms or frameworks, but few follow through with prototypes that demonstrate their real-world viability. Rare exceptions, such as the 1001 Nights system [90], show the potential of bridging this gap, but overall, the lack of deployable prototypes and industry-aligned evaluation pipelines has limited the adoption of PCG methods in mainstream development. Without tangible evidence of scalability and player experience benefits, much of the research remains confined to experimental contexts.

Finally, the quality of generated content remains a major bottleneck. While PCG can generate large volumes of content, the outputs often exhibit issues such as unstable structures, repetitive patterns, or structurally unstable configurations. This problem is especially visible in physics-based games like *Angry Birds*, where even small gaps or misalignments can destabilize entire levels. In many cases, ensuring quality requires significant post-processing or manual intervention, which undermines the intended efficiency of PCG. The rise of machine learning and LLM-based methods has improved diversity and adaptability, but challenges persist in maintaining reliability, stability, and structural reliability.

Together, these limitations highlight why further work is needed not just in generating content, but also in repairing, refining, and validating generated levels. Addressing these gaps will be essential to move PCG from academic novelty to practical game development pipelines.

2.6.1 PCG Challenges in Physics-Based Games

Across GANs, VAEs, and transformers, a common limitation is the absence of embedded physics reasoning during generation. These models excel at replicating visual and structural patterns but cannot guarantee that the generated levels will remain stable when simulated. This introduces several technical difficulties into the process of generating stable physics-based

levels:

- **Implicit physical constraints:** While human designers intuitively account for load-bearing structures, AI models often focus on visual plausibility rather than physical feasibility.
- **Complex stability dependencies:** Small local changes can produce cascading failures due to interconnected forces in the simulated environment.
- **Balancing stability and structural preservation:** Excessive stabilization may alter the original structural configuration of a level, while under-supported structures may collapse prematurely.

2.7 Level Repair in Procedural Content Generation

While most research in Procedural Content Generation (PCG) focuses on the generation phase, the repair of generated content remains comparatively underexplored, especially for physics-based games where stability is a critical requirement for structural reliability. Level repair refers to the process of detecting and correcting flaws in a generated level to improve its structural quality and stability without significantly altering its intended design.

2.7.1 Manual Repair Approaches

In industry settings, unstable or low-quality generated levels are often addressed through manual editing by designers. This process can involve:

- Adding or adjusting structural supports.
- Replacing misplaced or floating elements.
- Realigning components to improve load distribution.

While manual repair can produce structurally reliable outcomes, it is time-consuming, resource-intensive, and undermines one of the main goals of PCG, which is reducing human design effort.

2.7.2 Automated Repair Methods

To reduce reliance on manual intervention, several automated repair methods have been explored. These approaches vary by game type and repair objective:

- Constraint-based post-processing: levels are validated against predefined design rules, with violations automatically corrected [29].
- Search-based repair: iterative adjustments are made to structural configurations until evaluation criteria are satisfied.
- Machine learning-based repair: neural networks learn the probability distribution of tiles and replace unlikely or misplaced ones based on their surroundings [99].
- Computer vision-based repair: object detection or segmentation techniques are used to locate missing or misaligned elements.

These methods have shown promise in tile-based platformers, where level geometry is discrete and predictable. However, physics-based games such as *Angry Birds* pose unique challenges. Stability in such games emerges from continuous interactions between geometry, material properties (e.g., wood, ice, stone), and gravity. Unlike grid-based environments, a small misalignment or missing block can destabilize entire structures.

2.8 Object Segmentation

Object segmentation is a computer vision task that assigns a class label to each pixel in an image, enabling precise localization of objects and structural regions [100]. In our context,

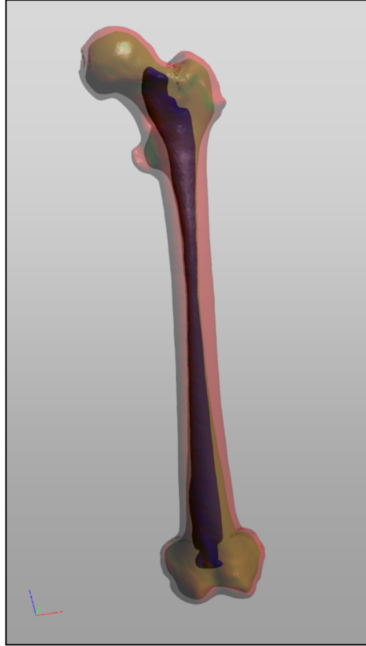


Figure 2.4: Segmented 3D model of a left human femur generated using MeVisLab, showing the outer surface (red), the interface between compact and spongy bone (green), and the bone marrow surface (blue).

segmentation enables the identification of structural gaps and discontinuities from an image-based representation of a game level. This section describes the segmentation architectures evaluated for automatic gap detection.

2.8.1 U-Net

U-Net, proposed by Ronneberger et al. [101], is one of the most influential architectures in image segmentation, particularly within biomedical imaging. Its design follows an **encoder-decoder paradigm**:

- The **encoder path** (or contracting path) consists of repeated convolutional and pooling layers that progressively downsample the input image, capturing increasingly abstract features while reducing spatial resolution.
- The **decoder path** (or expansive path) gradually upsamples the feature maps using transposed convolutions, reconstructing a segmentation mask that matches the input

resolution.

- **Skip connections** are introduced between encoder and decoder layers at the same resolution level. These connections pass high-resolution spatial features from the encoder directly to the decoder, ensuring that fine-grained details are preserved during reconstruction.

This symmetric U-shaped design allows U-Net to combine global context with local spatial precision, making it highly effective at pixel-level tasks such as gap detection. However, U-Net’s reliance on convolutional filters limits its ability to model long-range dependencies, which can be important in complex structural layouts.

2.8.2 SegFormer

SegFormer, introduced by Xie et al. [102], is a more recent architecture that leverages the power of transformers for semantic segmentation. Unlike convolutional models, transformers excel at modeling long-range dependencies and capturing both global and local context simultaneously. SegFormer achieves this through the following design principles:

- A **hierarchical transformer encoder** processes the input image in progressively smaller patches, allowing the model to capture both fine details and global structure. Unlike Vision Transformers (ViT), SegFormer uses *overlapping patch embeddings* to avoid losing spatial continuity at patch boundaries.
- A **Mix-FFN (Feed-Forward Network)** replaces standard position encodings, making the model position-agnostic while still able to capture relative spatial information. This improves generalization across varying input layouts.
- A lightweight **MLP decoder** aggregates multi-scale features from the encoder and upsamples them to produce a segmentation mask. The decoder is intentionally simple to keep inference efficient, in contrast to more complex CNN-based decoders.

By combining hierarchical feature extraction with efficient decoding, SegFormer balances accuracy and computational cost. Its strength lies in handling diverse structural patterns and capturing global relationships, which are valuable for detecting distributed gaps across an entire level layout.

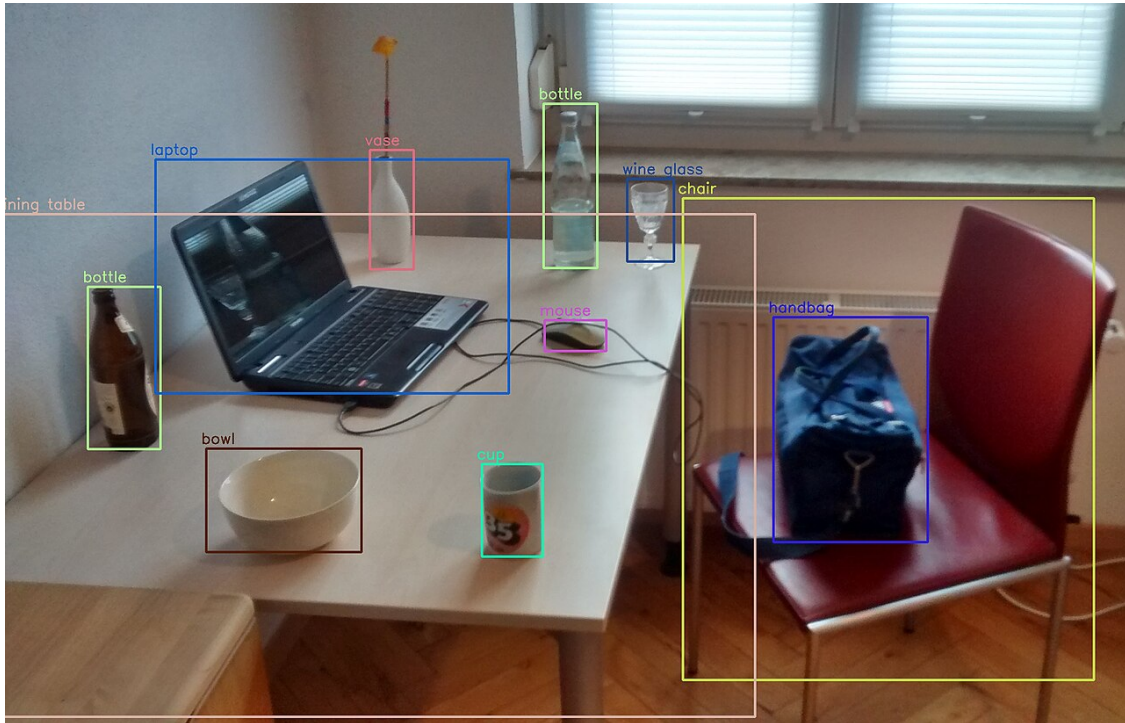


Figure 2.5: Objects detected using OpenCV’s Deep Neural Network (DNN) module with a YOLOv3 model trained on the COCO dataset. The model detects 80 common object categories.

2.8.3 YOLOv8m-Seg

YOLO (You Only Look Once), introduced by Redmon et al. [103], was the first real-time, end-to-end object detection framework, as shown in Figure 2.5. Its core innovation was reformulating detection as a single regression problem, allowing object locations and class probabilities to be predicted in one forward pass. This eliminated the need for computationally expensive sliding-window methods and multi-stage region proposal pipelines used in earlier detectors such as Fast R-CNN [104]. As a result, YOLO significantly improved inference

speed while maintaining competitive accuracy [105].

Modern YOLO architectures extend this framework to instance segmentation. In particular, YOLOv8-Seg [106] integrates a segmentation head that produces pixel-level masks alongside bounding box predictions. This dual representation is advantageous for gap detection, as bounding regions capture coarse structural context while segmentation masks provide fine-grained geometric localization.

YOLOv8 offers multiple model scales designed to balance accuracy and computational cost:

- **YOLOv8n-Seg (Nano):** Lightweight and optimized for speed, with reduced representational capacity.
- **YOLOv8s-Seg (Small):** A balanced variant offering improved accuracy while maintaining efficient inference.
- **YOLOv8m-Seg (Medium):** A larger model with greater representational power, enabling improved performance on complex and irregular structures.

We evaluated all three variants to assess the trade-off between model complexity and gap detection performance. Because structural gaps in unstable levels vary in scale and geometry, a model capable of capturing multi-scale spatial relationships is essential. The detailed results of these variants are presented in Chapter 4.

Chapter 3

Methodology

This chapter outlines our pipeline for the automated repair of AI-generated levels, using *Angry Birds* levels generated by the Angry Birds GAN model [18]. A typical *Angry Birds* level, as shown in Figure 3.1, contains a slingshot, birds, pigs, and a collection of blocks arranged in one or more structures [107]. While Procedural Content Generation (PCG) has shown remarkable progress in generating creative content, the issue of structural instability remains a bottleneck, particularly in physics-based games. Our methodology addresses this gap through a segmentation-driven repair pipeline that leverages computer vision techniques to detect structural gaps and applies automated corrections to stabilize levels.

3.1 Repairing Angry Birds Levels

Generative Adversarial Networks (GANs) and other AI-driven approaches have demonstrated the ability to create novel and varied *Angry Birds* levels. However, these levels frequently suffer from instabilities, including unsupported blocks, floating elements, and collapses immediately after loading into the game engine [18]. Such instabilities render levels structurally unreliable, significantly reducing their practical usability.

Instead of attempting to generate entirely stable levels from scratch, our approach focuses on **repairing unstable levels post-generation**. This strategy is motivated by the



Figure 3.1: Screenshot of a level from the *Science Birds* framework, a research platform based on the gameplay rules and physics mechanics of the commercial game *Angry Birds*.

observation that many generated levels are *nearly stable* but fail due to localized weaknesses. By detecting and correcting these weak points, we can salvage otherwise unusable levels, thus increasing the effective yield of PCG systems.

3.2 Model Architecture Selection

For the gap detection task, we selected three distinct, state-of-the-art segmentation model architectures for comparison: U-Net, SegFormer, and YOLOv8m-Seg, the medium-capacity variant of the YOLOv8 segmentation architecture. This selection was made to evaluate a representative cross-section of modern segmentation paradigms. U-Net was chosen as a highly effective and well-established benchmark for convolutional-based segmentation. SegFormer represents the more recent transformer-based approach, which has achieved strong performance in various computer vision tasks. Finally, YOLOv8m-Seg was included for its foundation in real-time object detection, offering a computationally efficient alternative. By

comparing these three, we aimed to identify the most suitable architecture balancing accuracy and performance for our level repair pipeline.

3.2.1 Comparison of the Three Architectures

Each model offers complementary advantages:

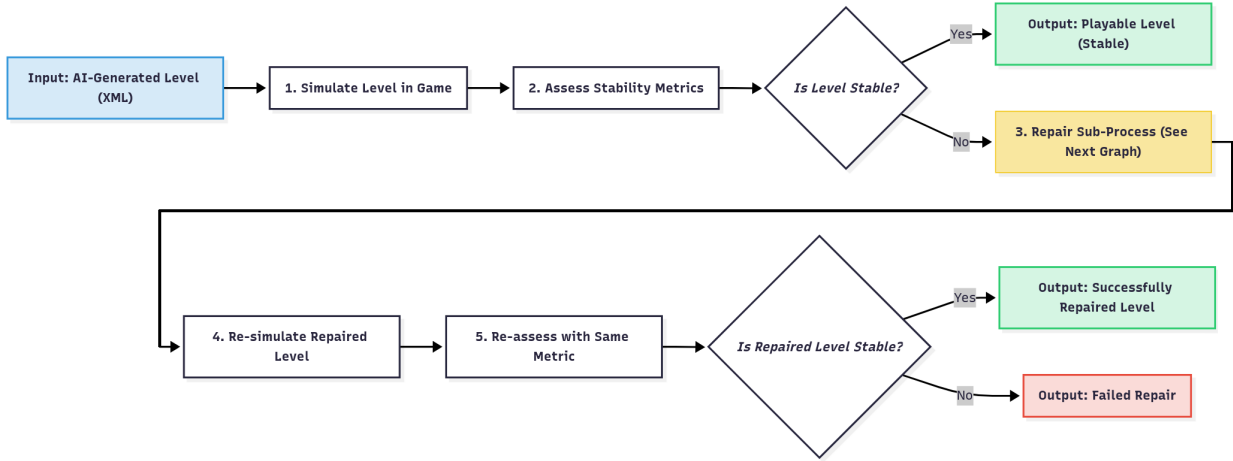
- U-Net provides strong pixel-level accuracy and is effective on smaller datasets.
- SegFormer captures global structural patterns and generalizes well to varied configurations.
- YOLOv8m-Seg balances segmentation accuracy with high-speed inference, making it practical for large-scale automated repair.

Through comparative evaluation (see Chapter 4), YOLOv8m-Seg achieved the strongest overall segmentation performance while maintaining efficient inference speed. Based on this balance between accuracy and computational cost, YOLOv8m-Seg was selected as the backbone of the repair framework.

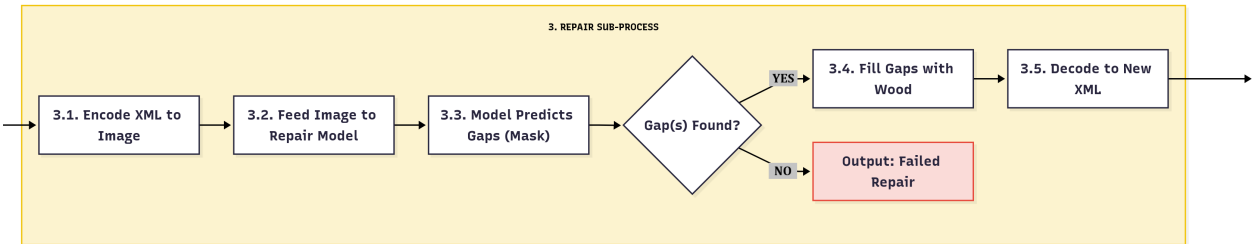
3.3 Dataset

For training, Abraham and Stephenson used the open-source level generator Iratus Aves [27] to produce a dataset of 4,931 level descriptions, each representing a unique structure, including pigs and a variety of blocks. We used this dataset exclusively for training and validation of the segmentation models.

Given our focus on teaching the model to identify gaps that destabilize structures, we filtered these levels to select only those that were stable. This was essential because our goal was to train a model on de-stabilized versions of otherwise stable levels, enabling supervised learning with known ground-truth gaps that could be filled to restore stability.



(a) Repair Pipeline



(b) Repair Sub-Process

Figure 3.2: The automated level repair pipeline. The top diagram (a) illustrates the high-level workflow, where an AI-generated level is first simulated and evaluated for stability. If the level is deemed unstable, it enters the repair stage. After the repair is applied, the level is re-evaluated to confirm its stability. The bottom diagram (b) provides a detailed breakdown of the “Repair Sub-Process” block, showing the five steps from encoding the level into an image, using the model to predict gaps, filling the gaps with wood, and decoding the result back into a new XML file.

We evaluated level stability using three different metrics derived from physical simulation. To ensure greater consistency and reliability in training, we selected the most stringent stability metric, as it yielded the most robust set of stable levels. This filtered set became our stable dataset, and we reduced the level count to 1,887 XML levels.

To generate training samples with structural gaps, we artificially introduced instability by removing one to four random blocks from the XML file of each selected level. We then simulated these modified levels, keeping only those where the original level was stable but the modified version became unstable after block removal. Out of the 1,887 stable levels,

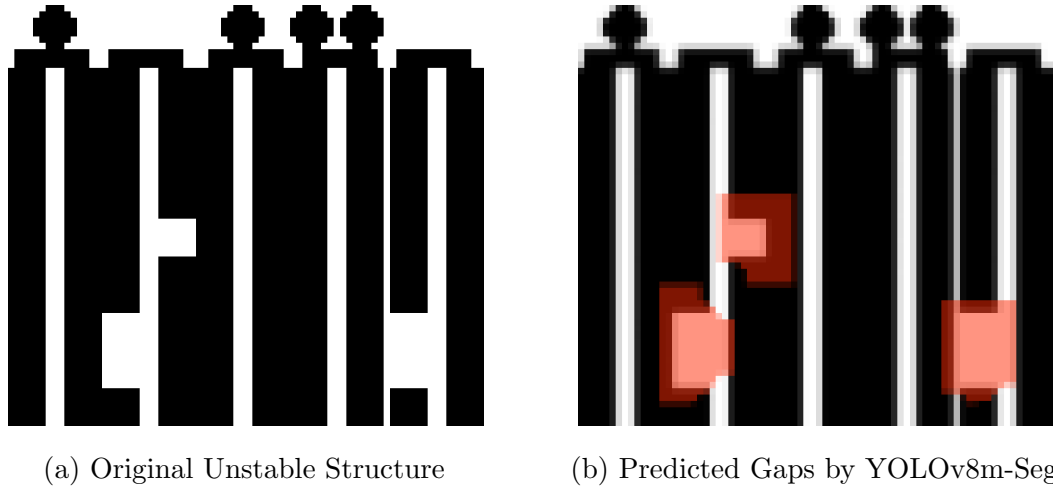


Figure 3.3: A visual demonstration of the YOLOv8m-Seg model’s gap detection capability. (a) The binary image of an unstable level is given as input. (b) The model outputs a segmentation mask (shown as red overlays) that identifies the location and shape of structural gaps.

1,547 could be successfully destabilized through controlled block removal while preserving meaningful structural configurations. These levels constituted the final dataset used for training.

For model training, each level’s XML file was processed using the established encoding pipeline. The level was converted into a multi-layer grid representation, which was then flattened into a single 2D image. To focus the model on structure rather than material composition, this image was converted into a binary format (where 1 represents any object and 0 represents empty space). This binary image of the unstable structure served as the input for our model, while the pixel-perfect location of the removed block(s) served as the target segmentation mask.

This procedure ensured that each training sample corresponded to specific and known destabilizing gaps, aligning with our goal of using supervised learning to detect and suggest gap-filling interventions.

For evaluation of the full repair pipeline, we used the complete set of 8,000 GAN-generated *Angry Birds* levels reported by Abraham and Stephenson in their original study. These levels correspond to the full output of the GAN-based generator described and analyzed in their

paper and were not used at any stage during training or validation. Instead, they were reserved exclusively as unseen test data to evaluate the repair pipeline’s effectiveness and its ability to generalize to novel, previously unobserved structures. This dataset was selected for two primary reasons. First, the authors noted significant instability issues within their generated levels, which provided an ideal test case for our repair pipeline. Second, using their established dataset allows for a more direct comparison of the improvements offered by our method.

We acknowledge that this design may not capture all possible structural failure modes. These limitations are discussed further in the Limitations section.

3.4 Training Details

The training of our models and the repair process were conducted on a single local workstation.

3.4.1 Training Configuration

Each segmentation model (U-Net, SegFormer, YOLOv8m-Seg) was trained under a consistent experimental setup to enable fair comparison. All models were trained for a maximum of 100 epochs with an early stopping criterion (patience = 10) based on loss to avoid overfitting.

Training was conducted on a local Windows-based machine with the following specifications: CPU: Intel Core™ i5-10400F, GPU: NVIDIA GeForce GTX 1660 Super, and 16 GB RAM. A batch size of 8 was used for all models.

Input images were resized to 128×128 pixels to balance computational efficiency with segmentation performance. The dataset was split into 80% training and 20% validation sets.

The AdamW optimizer was used with default β parameters (0.9, 0.999). Model weights were saved every 5 epochs, and the best-performing checkpoint based on validation F1-score was selected for evaluation in the repair pipeline.

To ensure reproducibility, random seeds were fixed across NumPy, PyTorch, and CUDA

libraries. This standardized configuration ensured that performance differences between U-Net, SegFormer, and YOLOv8m-Seg were attributable to architectural differences rather than variations in training procedure.

Upon completion of training, the models were compared based on their segmentation performance on the validation dataset. To measure the precise quality of the generated masks against the ground truth, we calculated several pixel-level metrics such as Precision, Recall, and F1-Score for each model.

3.5 Repair Process

The automated repair pipeline is the core component of our methodology, transforming unstable AI-generated levels into structurally stable configurations. This process follows a three-stage workflow of evaluation, repair, and re-evaluation. The workflow is designed to minimize false positives, preserve the original structure as much as possible, and provide an efficient mechanism for stabilizing levels at scale. Figure 3.2 illustrates the overall design.

3.5.1 Stage 1: Stability Evaluation

Each level generated by the GAN is first loaded into the *Science Birds* physics engine and simulated for a fixed duration without any player input. This step determines whether the structure remains intact under gravity or exhibits signs of instability.

The evaluation is based on three quantitative metrics:

- **Block Velocity (BV):** The mean absolute velocity of blocks immediately after loading. Levels with low BV values are considered stable, while higher values indicate collapse or motion.
- **Block Destruction (BDes):** The percentage of blocks destroyed during simulation. A non-zero destruction rate indicates significant instability.

- **Block Damage (BD):** The cumulative damage sustained by all blocks. This captures cases where the structure partially shifts or deforms without full destruction.

Only levels classified as unstable by at least one of these metrics are forwarded to the repair module. Stable levels bypass this process, preventing unnecessary modifications.

3.5.2 Stage 2: Automated Repair Operations

Once a level is flagged as unstable, it undergoes the repair stage, which leverages the trained segmentation model to identify gaps and weak regions.

Binary Encoding and Segmentation

The XML description of the unstable level is first converted into a binary image representation, where occupied pixels represent blocks and empty pixels represent open space. This abstraction reduces the complexity of material-specific physics and allows the segmentation model to focus purely on structural geometry.

The binary image is passed to the trained segmentation model (YOLOv8m-Seg), which outputs a gap mask. The mask highlights contiguous regions where missing or misaligned blocks are likely to be the cause of instability.

Gap Mask Processing

The raw segmentation mask is refined through several steps:

1. Thresholding is applied to eliminate low-confidence predictions.
2. Morphological operations (dilation and erosion) are performed to smooth boundaries and remove isolated noise pixels.
3. Connected component analysis groups pixels into coherent gap regions, each treated as a candidate for repair.

Block Placement Strategy

For each detected gap, the repair module inserts an appropriately sized block:

- **Material Selection:** Wood is chosen as the default repair material. It provides balanced durability, moderate weight, and predictable interactions. Stone, while stronger, risks introducing new collapses due to excess weight. Ice, on the other hand, is too fragile and slippery, often reducing stability. Our preliminary tests confirmed that the choice of material has minimal impact compared to block placement, justifying the default use of wood.
- **Shape and Size:** The closest matching shape from the Science Birds block library is selected (e.g., RectSmall, SquareTiny). This ensures compatibility with the game’s predefined physics assets.
- **Placement Coordinates:** Gap regions are aligned with gravity and anchored to adjacent stable structures. Misalignment is avoided by snapping placement coordinates to the nearest block edges where possible.

3.5.3 Stage 3: Re-evaluation of Repairs

The final stage converts the modified binary image back into the standard XML format used by the Science Birds engine. The repaired level is then simulated again under the same stability metric that originally flagged it as unstable. This ensures consistency between detection and validation.

If the level meets the stability thresholds (low BV, minimal BD, and zero or near-zero BDes), it is marked as successfully repaired and added to the pool of stable levels. If not, the level is classified as unsuccessfully repaired, but its stability metrics are still recorded for analysis. Interestingly, in many cases, even unsuccessful repairs reduced the severity of collapse by lowering block damage or destruction rates.

3.5.4 Design Choices and Rationale

Several design decisions were made to balance accuracy, efficiency, and structural preservation:

- **Default material as wood:** Chosen for stability without introducing excess weight or fragility.
- **Binary image representation:** Simplifies the problem to geometry rather than material physics, improving model generalization.
- **Localized repairs only:** To preserve the original structural design, only detected gaps are modified. Over-repair is avoided to prevent unnecessary alteration of level configurations.
- **Iterative verification:** The same evaluation metrics are used before and after repair, ensuring consistency in classification.

3.5.5 Illustrative Example

Figure 3.4 demonstrates a complete cycle: (a) an unstable GAN-generated level is flagged during simulation, (b) the segmentation model predicts structural gaps, and (c) new blocks are inserted to stabilize the structure. After re-simulation, the repaired level remains intact, confirming the success of the repair process.



(a) Initial Unstable Level



(b) Simulation Result of Unstable Level



(c) Repaired Stable Level

Figure 3.4: Illustration of a successful level repair: (a) The AI-generated level is flagged as unstable. (b) The physics simulation confirms instability as the structure collapses. (c) After our system repairs the level, it becomes stable.

Chapter 4

Evaluation and Results

This chapter presents a comprehensive evaluation of our proposed level repair pipeline. The evaluation is divided into two main components. First, we conduct a comparative analysis of three segmentation model architectures—U-Net, SegFormer, and YOLOv8m-Seg—to determine the most suitable backbone for our gap-detection task. Second, we evaluate the complete repair pipeline, using the best-performing model, to measure its impact on stabilizing AI-generated levels from Abraham and Stephenson’s dataset [18].

The central aim of this evaluation is to address the following research questions:

1. Which segmentation architecture provides the best trade-off between accuracy and efficiency for detecting structural gaps in *Angry Birds* levels?
2. To what extent can the proposed repair pipeline increase the number of stable levels?
3. What are the characteristics of unsuccessful repairs, and do they still provide a partial stabilizing effect?

4.1 Evaluation Metrics

We employ two broad categories of metrics: (1) *model performance metrics*, which measure the accuracy of gap segmentation at the pixel level, and (2) *level stability metrics*, which

assess the efficacy of the repair pipeline in producing structurally sound levels inside the game environment. By combining both categories, we capture not only the predictive accuracy of the models but also their practical impact on structural stability.

4.1.1 Model Performance Metrics

To evaluate the segmentation quality of U-Net, SegFormer, and YOLOv8m-Seg, we use standard pixel-level classification metrics that measure how closely the predicted segmentation masks match the ground-truth annotations:

- **Precision:** The proportion of predicted gap pixels that were correctly classified. High precision indicates that the model avoids false positives (unnecessary repairs).
- **Recall:** The proportion of true gap pixels correctly identified. High recall ensures that most structural weaknesses are detected, reducing the chance of leaving critical gaps unaddressed.
- **F1-Score:** The harmonic mean of Precision and Recall, providing a single balanced measure of segmentation quality.
- **Loss:** The optimization objective used during training to quantify the difference between predicted and ground-truth segmentation masks. It is minimized by the model to improve prediction accuracy. Lower loss values indicate better alignment between predicted gap regions and the ground truth.

Figure 4.1 summarizes the training behavior of the YOLOv8m-Seg, SegFormer, and U-Net models, showing their F1 score, precision, recall, and loss over training epochs.

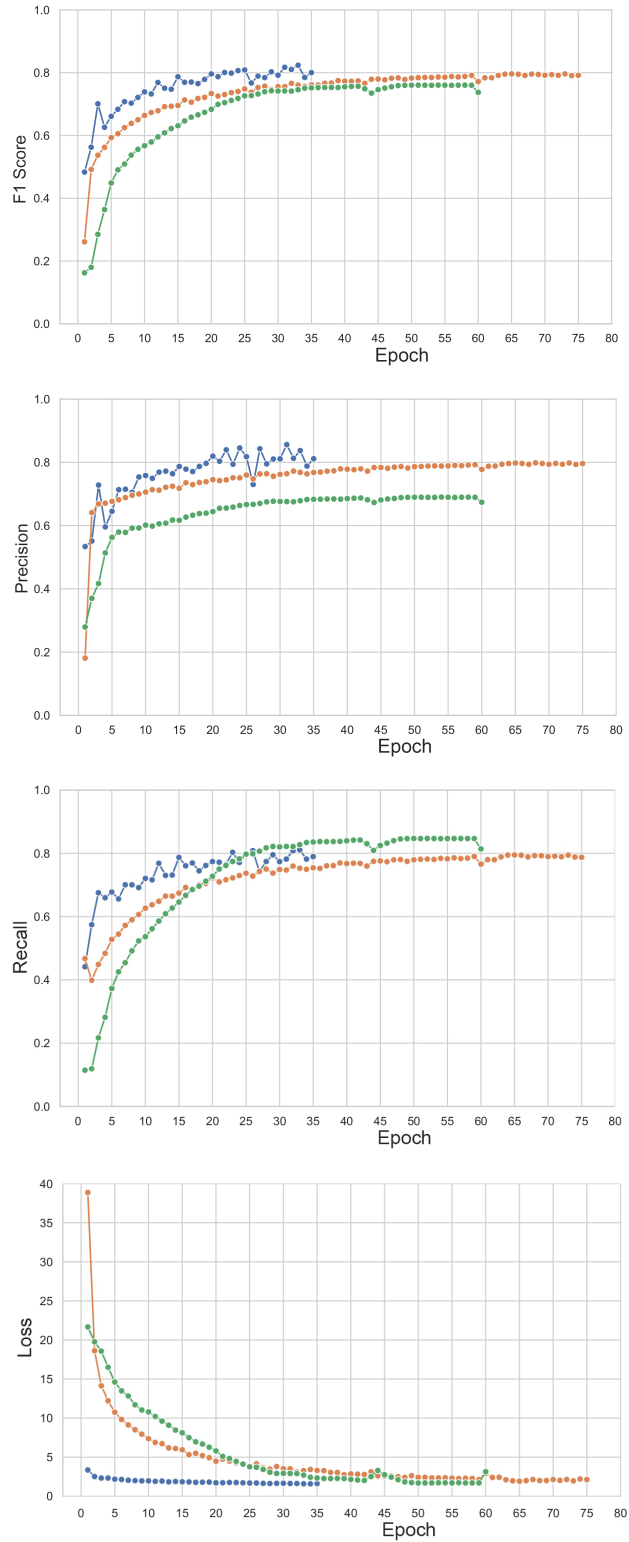


Figure 4.1: Training Progress of YOLOv8m-Seg (blue), SegFormer (orange), and U-Net (green) models over epochs.

While these metrics provide a strong baseline for comparison, they are primarily pixel-focused and may not fully capture the quality of structural localization. For this reason, we also report object detection-inspired metrics for YOLO variations, which emphasize the accuracy of spatial localization:

- **Intersection over Union (IoU):** Measures the ratio of overlap between the predicted mask and the ground-truth mask to the area of their union. IoU provides a more spatially aware evaluation than pixel-level accuracy alone.
- **Mean Average Precision (mAP):**
 - **mAP@50:** Evaluates precision at a fixed IoU threshold of 0.50, capturing how well the model detects gaps at a moderate overlap requirement.
 - **mAP@50–95:** Averages precision across IoU thresholds from 0.50 to 0.95 in increments of 0.05. This stricter variant rewards models that produce highly precise masks closely aligned with ground truth, rather than simply coarse detections.

The inclusion of IoU and mAP complements the pixel-based metrics by assessing not only whether a gap was identified but also how accurately its shape and boundaries were localized, which is critical for effective repairs.

4.1.2 Level Stability Metrics

To evaluate the practical impact of the repair pipeline, we measure the stability of levels using three distinct metrics. These metrics, adapted and extended from prior work [18], capture different facets of structural stability, ranging from strict physical constraints to more lenient definitions of acceptable structural behavior:

- **Block Velocity (BV):** A level is considered stable if all blocks are stationary after loading. This is the strictest metric, as even minor movements or shifts disqualify a level from being considered stable. It serves as a strong indicator of structural soundness.

- **Block Destruction (BDes):** A level is considered stable if no blocks are destroyed during the initial simulation. This metric captures catastrophic structural failures (e.g., block collapse) but is more lenient than BV since small displacements without destruction are tolerated.
- **Block Damage (BD):** A level is considered stable if the cumulative block damage, as quantified by the game’s physics engine, is less than or equal to zero (the default intact value). This metric captures intermediate cases where blocks sustain damage or deformation without being destroyed. It provides a more nuanced perspective that distinguishes between trivial and severe instabilities.

Together, these three metrics form a multi-level framework for quantifying stability:

1. **Strict (BV):** Ensures full immobility and structural integrity.
2. **Intermediate (BD):** Allows for minor damages while filtering out severe degradation.
3. **Lenient (BDes):** Focuses on the absence of catastrophic destruction, permitting small shifts and tolerable damage.

This tiered evaluation is particularly valuable because not all instabilities affect structural behavior equally. For example, a slight block displacement (BV violation) may indicate minor instability, whereas destruction (BDes violation) reflects a more severe structural failure. By using all three metrics, we ensure that our evaluation captures both fine-grained physics behavior and the practical structural stability of repaired levels.

4.2 Comparing Model Architectures

The comparative evaluation results are shown in Table 4.1.

The results highlight key trade-offs between the three architectures:

Model	Epoch	Precision	Recall	F1 Score	Loss
YOLOv8m-Seg	33	0.837	0.811	0.824	1.596
SegFormer	65	0.798	0.795	0.796	1.932
U-Net	54	0.690	0.847	0.760	1.721

Table 4.1: Performance comparison of three segmentation models at their best epochs, evaluated using precision, recall, F1-Score, and training segmentation loss.

- YOLOv8m-Seg:** YOLOv8m-Seg demonstrates the strongest overall performance, achieving the highest F1-Score (0.824) and the lowest segmentation loss (1.596). Its balanced precision (0.837) and recall (0.811) indicate that it can reliably identify most gaps while avoiding unnecessary over-repairs.
- SegFormer:** SegFormer produced consistent results, with an F1-Score of 0.796 and nearly balanced precision (0.798) and recall (0.795). Its higher number of training epochs to reach convergence (65) reflects the heavier computational cost of transformer-based models compared to convolutional or hybrid approaches. However, its strength lies in its ability to generalize well to diverse and unseen layouts due to the transformer’s capacity for modeling long-range dependencies. This makes SegFormer a viable option for applications where level designs are highly varied, or where global spatial reasoning is particularly important.
- U-Net:** U-Net achieved the highest recall (0.847), indicating strong sensitivity in detecting nearly all true gaps. However, this came at the expense of precision (0.690), meaning that it frequently predicted gap regions where none existed. This over-repair behavior often resulted in excessive block placement, which, while improving stability, risked altering the original structural configuration of levels. Despite this limitation, U-Net’s simple and lightweight design remains advantageous for rapid prototyping and as a baseline model in segmentation-driven repair tasks.

Beyond pixel-level metrics, YOLOv8m-Seg also achieved strong object-detection metrics (mAP@50 = 0.862 and mAP@50–95 = 0.409), reflecting its superior ability to localize

gaps with accurate boundaries. This ability is critical in the repair process, where small misalignments can propagate into structural instabilities. Importantly, YOLO’s real-time inference speed makes it well-suited for deployment in automated repair pipelines or industry-level quality assurance workflows.

4.2.1 Interpretation of Trade-offs

The comparison illustrates clear architectural trade-offs:

- U-Net favors recall over precision, maximizing detection but risking excessive repairs.
- SegFormer offers balanced precision and recall with strong generalization but at higher computational cost.
- YOLOv8m-Seg balances both precision and recall while excelling in spatial localization, providing the best trade-off between accuracy and efficiency.

4.2.2 Implications for the Repair Pipeline

These findings justify selecting YOLOv8m-Seg as the backbone of our repair framework. Its superior F1-Score, precise localization make it the most practical architecture for automated repair, where both accuracy and speed are critical. While SegFormer and U-Net remain viable alternatives depending on design constraints (e.g., variety of levels or computational budget), YOLOv8m-Seg provides the strongest combination of stability improvements and accurate localization, aligning best with the requirements of large-scale industry adoption.

Beyond the primary metrics reported in Table 4.1, we conducted a focused comparison of YOLOv8 segmentation variants (YOLOv8n-Seg, YOLOv8s-Seg, and YOLOv8m-Seg) to further analyze performance trade-offs across detection and segmentation objectives.

In addition to precision, recall, and F1-Score, we evaluated mean Average Precision (mAP) and loss components (segmentation, classification, and bounding box losses). We also computed aggregated measures such as the global harmonic mean across bounding box and

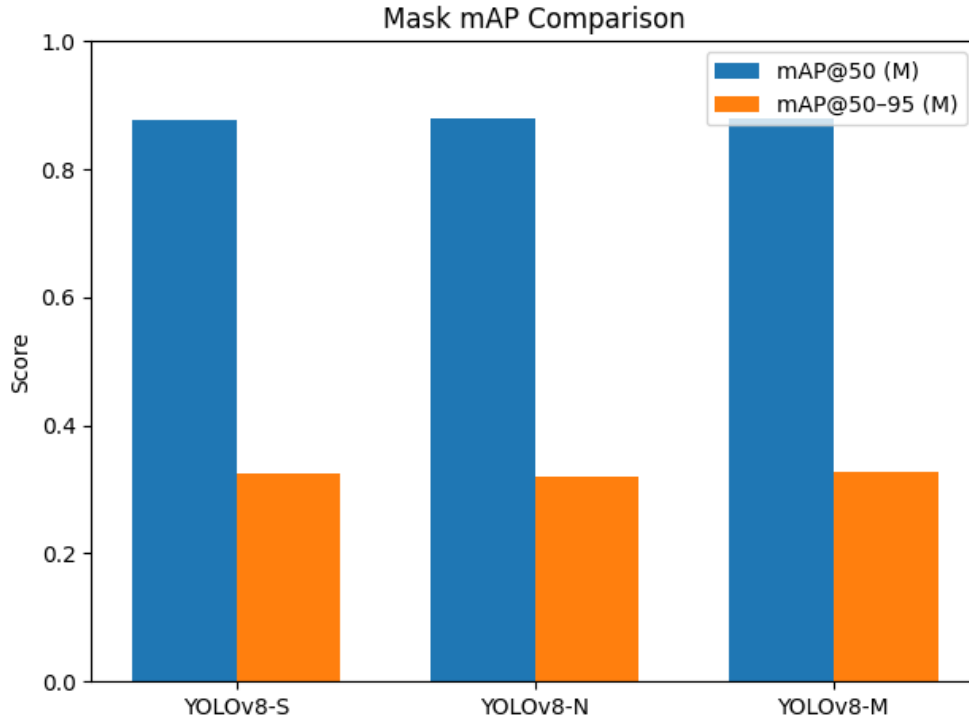


Figure 4.2: Comparison of mask-based mean Average Precision (mAP) across YOLOv8 variants.

mask predictions. Figure 4.2 presents a comparative bar chart of mask-based mAP scores (mAP@50 and mAP@50-95) across the three YOLOv8 variants.

Across all metrics, **YOLOv8m-Seg consistently outperformed the smaller variants**, achieving the highest F1-Score (0.824), the strongest mAP@50-95 values, and the best overall harmonic mean. These results further support the selection of YOLOv8m-Seg as the most balanced and reliable model for gap detection in the repair pipeline.

4.3 Efficacy of the Repair Pipeline

4.3.1 Quantitative Repair Success

We applied the repair pipeline, powered by YOLOv8m-Seg, to the pool of unstable levels. Results are summarized in Table 4.2.

Key observations:

Metric	Initial Unstable	Stabilized	Repair Rate	Initial Stable	Final Stable	Growth Factor
Block Velocity	7,055	1,254	17.8%	945	2,199	2.33
Block Damage	6,259	1,452	23.2%	1,741	3,193	1.83
Block Destruction	4,533	2,051	45.3%	3,467	5,518	1.59

Table 4.2: Summary of Repair Outcomes Across Three Stability Metrics.

- Under the strictest metric (Block Velocity), our method doubled the number of stable levels (Growth Factor = 2.33). This demonstrates that the pipeline is effective even under conservative definitions of stability where any movement is disallowed.
- Under Block Damage, repair yielded a Growth Factor of 1.83, stabilizing over 1,400 additional levels. This reflects the pipeline’s ability to mitigate partial instabilities, preventing cumulative damage without necessarily achieving perfect stillness.
- The lenient Block Destruction metric shows the highest Repair Rate (45.3%), but a smaller Growth Factor (1.59), reflecting its already high baseline of stable levels. This suggests that repairs were particularly impactful for borderline-unstable structures.

Overall, these results highlight that while absolute repair success varies depending on the stability definition, our pipeline consistently increases the pool of structurally stable levels, effectively turning previously unstable content into viable game assets.

4.3.2 Analysis of Unsuccessful Repairs

Not all repairs succeeded. We categorize the failures into two modes:

1. **No gaps detected:** In approximately 24% of failed cases, the model did not detect structural gaps. These instabilities were likely caused by other factors such as poor weight distribution, weak support chains, or overly top-heavy designs—issues beyond the scope of segmentation-driven repair.

2. **Partial repair:** In other cases, gaps were correctly detected and filled, but the intervention was insufficient to prevent collapse. This was especially common in multi-gap structures, where filling only one or two voids did not provide enough reinforcement to stabilize the level. Severe instabilities with cascading block failures also remained unresolved.

Despite these failures, partial mitigation was observed. Compared to unrepaired unstable levels, failed repairs still reduced average block damage by 25.1% and average destroyed blocks by 11.5%. This indicates that even when full stabilization was not achieved, the pipeline still contributed to improved structural integrity in levels that would otherwise have collapsed completely.

Chapter 5

Discussion

5.1 Interpretation of Findings

The results presented in Chapter 4 confirm that the proposed segmentation-driven repair framework substantially improves the stability of AI-generated *Angry Birds* levels. This chapter interprets these findings in light of the research objectives outlined in Chapter 1 and situates them within the broader context of procedural content generation (PCG), automated repair, and industry-level game design practices.

The discussion is structured around four core themes. First, we examine the comparative performance of the segmentation models, highlighting the architectural strengths and trade-offs of YOLOv8m-Seg, SegFormer, and U-Net. Second, we consider the implications of repair strategies on structural properties of levels, analyzing the extent to which stability improvements affect structural consistency and stability characteristics. Third, we explore the applicability of the framework in professional game development pipelines, identifying potential integration pathways as a designer-assist tool, a post-processing module, or an automated quality assurance system. Finally, we outline the current system’s limitations, including model dependencies and game-specific tuning requirements, before considering future directions that could broaden the framework’s generalization and adaptability.

The model is generalizable because it learns structural patterns from an image representation rather than relying on game-specific rules. Since the input is a binary spatial representation, the model focuses on geometric relationships like gaps, support, and connectivity. These patterns are not unique to *Angry Birds* and can appear in other structured environments, which is why the approach can transfer in principle.

By integrating these perspectives, the discussion highlights not only the empirical success of the proposed repair pipeline but also its potential to serve as a scalable and practical solution for the games industry. At the same time, it acknowledges the challenges and design trade-offs that must be addressed to ensure the framework can generalize to different genres, adapt to different structural requirements across game environments, and operate efficiently at production scale.

5.1.1 Impact of Repairs on Structural Properties

One of the key contributions of this research is the demonstration that stability improvements can be achieved while preserving the original structural characteristics of a level. Manual evaluation conducted by the author indicated that repaired levels were generally perceived as more stable, particularly when the repairs were subtle and blended naturally into the original design. This highlights the importance of perceptual coherence: repairs that appear as natural extensions of the existing structure preserve the original structural design of the level generator.

At the same time, the findings reveal that not all repair outcomes were equally beneficial. In particular, excessive reinforcement had the side effect of altering the original structural behavior. By closing too many gaps or inserting more blocks than necessary, these repairs sometimes produced overly stable structures, which reduced the structural variability of *Angry Birds* levels. This reduced the diversity of structural outcomes observed across repaired levels.

These results suggest that repair algorithms must carefully balance two competing goals:

(1) maximizing physical stability, and (2) preserving the original structural properties of the level. An optimal repair system should aim not for maximum reinforcement, but for minimal and well-targeted interventions that stabilize a level just enough for it to remain structurally stable while maintaining its original configuration. Future iterations of this framework could integrate difficulty-preservation heuristics or adaptive reinforcement thresholds to better maintain this balance.

5.1.2 Industry Applicability

The proposed repair framework holds promise for integration into professional game design pipelines, where stability, scalability, and efficiency are critical. We identify three main modes of integration:

1. As a **post-processing module** for AI-generated levels, where every newly generated level is automatically checked and repaired prior to release. This ensures that structurally unstable levels are filtered out, improving overall content quality with minimal additional cost.
2. As a **designer assist tool**, embedded within level editors or game design environments. In this mode, the framework could present suggested repairs directly to human designers, who would then retain the final decision-making authority. This not only saves time for designers by automating tedious repair tasks but also fosters a human–AI collaborative workflow where designers can refine or override machine suggestions.
3. As part of an **automated QA system** for large-scale content production pipelines. In contexts where thousands of levels are generated or updated dynamically, manual verification is infeasible. Integrating this repair framework into QA systems enables scalable, automated validation and stabilization, ensuring that content meets quality standards before distribution.

Across these modes, the framework’s efficiency and accuracy make it well-suited for industry adoption. Of the evaluated architectures, YOLOv8m-Seg paired with the repair module emerged as the most practical configuration, offering both high precision in detecting gaps and fast inference suitable for real-time or batch-processing environments. Its deployment could reduce the burden on designers, ensure consistent structural stability across generated levels, and ultimately accelerate the integration of procedural content generation into mainstream game development workflows.

Despite being trained and evaluated on a single consumer-grade workstation (Intel i5-10400F, GTX 1660 Super, 16 GB RAM), the proposed framework achieved strong performance. In contrast, the baseline work by Abraham and Stephenson [18] relied on high-performance computing resources (e.g., RWTH Aachen High-Performance Computing cluster). This highlights the computational efficiency of our approach, demonstrating that comparable stability improvements can be achieved without reliance on large-scale computing infrastructure.

5.2 Relation to Research Objectives

This section revisits the research objectives outlined in Chapter 1 and evaluates the extent to which they were achieved through the methods, experiments, and results presented in this thesis. By systematically aligning findings with the original aims, we provide a structured account of the framework’s contributions, strengths, and areas that warrant further refinement.

Objective 1: Develop a framework to detect and repair structural weaknesses in AI-generated game levels.

The proposed segmentation-driven repair framework successfully identified structural gaps in unstable levels generated by the GAN-based method of Abraham and Stephenson [18]. Three different segmentation architectures were integrated into the pipeline, and the repaired outputs were quantitatively evaluated using physics-based stability metrics. Across Block

Velocity, Block Damage, and Block Destruction evaluations, the framework consistently improved stability, in some cases more than doubling the pool of stable levels. Importantly, manual evaluation conducted by the author indicated that repaired levels preserved structural coherence.

Objective 2: Compare the performance of multiple segmentation architectures for gap detection.

A comprehensive experimental comparison of U-Net, SegFormer, and YOLOv8m-Seg was conducted. These architectures were selected to represent three distinct design paradigms: convolutional encoder–decoders (U-Net), transformer-based global attention (SegFormer), and real-time detection–segmentation hybrids (YOLOv8m-Seg). The models were benchmarked using Precision, Recall, F1-score, IoU, and loss. Results showed that YOLOv8m-Seg provided the strongest balance of localization accuracy, efficiency, and stability outcomes, while SegFormer demonstrated robust generalization and U-Net provided high recall but at the cost of over-repair. This systematic comparison highlights how architectural choices shape the success of automated level repair systems.

Objective 3: Ensure repairs preserve structural characteristics of levels.

Manual evaluation conducted by the author provided encouraging evidence that the repaired levels preserved structural coherence in most cases. Levels repaired by YOLOv8m-Seg and SegFormer were generally assessed more favorably, particularly when the repairs blended seamlessly with the original design intent. However, U-Net’s tendency toward over-repair occasionally altered the original structural configuration of levels more noticeably than the other models. This outcome emphasizes the importance of controlling the extent of structural modifications during repair: balance and challenge must also be preserved. Future work will

need to incorporate difficulty-aware repair strategies or adaptive thresholds to mitigate this limitation.

Objective 4: Evaluate the industry applicability of the repair framework.

The system was explicitly designed with production pipelines in mind. Three potential integration modes were identified: as a post-processing module ensuring stability in AI-generated levels, as a designer-assist tool providing suggested repairs for human approval, and as an automated QA mechanism for large-scale game content production. The efficiency and accuracy of YOLOv8m-Seg make it particularly well-suited for such contexts, with the potential to reduce manual QA costs while preserving creative flexibility. These findings confirm the framework’s practical viability and open the door for its adoption in industry-scale pipelines.

5.3 Limitations

- **Restricted Repair Strategy:** Repairs in the current framework are constrained to filling gaps with predefined block shapes and a fixed material (wood). While this ensures consistency and reduces complexity, it also limits the creativity and diversity of possible repairs. More flexible repair strategies—such as combining multiple materials, varying block orientations, or introducing novel structural patterns—could yield more optimized or aesthetically pleasing results that go beyond mere stabilization.
- **Segmentation Dependency:** The overall success of the pipeline is directly tied to the accuracy of the segmentation model. Mis-segmented gaps can lead to misplaced or unnecessary repairs, which in some cases may worsen stability rather than improve it. This dependency highlights the importance of continual improvements in segmentation architectures, training datasets, and post-processing filters to ensure reliable detection.

Objective	Description	Status	Supporting Evidence
1	Develop a framework to detect and repair structural weaknesses in AI-generated game levels	Achieved	Framework stabilized levels across all three metrics (BV, BD, BDes); demonstrated in repair pipeline results (Chapter 4).
2	Compare the performance of multiple segmentation architectures for gap detection	Achieved	Experimental comparison of U-Net, SegFormer, and YOLOv8m-Seg; metrics include Precision, Recall, F1-score, IoU (Chapter 4).
3	Ensure repairs preserve structural characteristics of levels	Partially Achieved	Manual evaluation conducted by the author indicated that repairs generally preserved structural coherence; U-Net over-repair highlighted the need for adaptive thresholds (Chapters 4–5).
4	Evaluate the industry applicability of the repair framework	Achieved	Integration pathways proposed: post-processing, designer-assist, QA automation; YOLOv8m-Seg identified as most deployable (Chapter 5).

Table 5.1: Mapping of research objectives to results, indicating achievement status and supporting evidence.

- Game-Specific Calibration:** Stability thresholds, material properties, and block physics were tuned specifically for the *Angry Birds* environment. While effective in this context, these assumptions may not transfer seamlessly to other physics-based games. For example, differences in gravity models, collision handling, or friction coefficients would likely require recalibration of stability metrics and material properties.
- Over-Repair Risks:** Excessive reinforcement can alter the original structural behavior of levels. This points to a broader consideration: repairs must not only achieve structural stability but also respect the game designer’s intent. Without such safeguards, there is a risk of producing levels that are technically stable but aesthetically or experientially unsatisfying.
- Computational and Integration Costs:** Although YOLOv8m-Seg provides relatively

efficient inference, training segmentation models and applying repairs at scale remains computationally demanding. In a production setting, this introduces integration challenges such as GPU availability, pipeline latency, and the need for batch-level optimizations to handle thousands of levels in real time or near real time.

- **Fixed Input Resolution:** All models were trained on images resized to 128×128 . While this choice improved training efficiency and reduced computational cost, it may result in the loss of fine-grained structural details, particularly for small gaps or thin support elements. Using higher-resolution inputs could improve segmentation precision, but would introduce additional computational overhead and longer training times.
- **Limited Domain Evaluation:** The framework was evaluated exclusively on *Angry Birds*-style levels. While the proposed approach is, in principle, generalizable, since it operates on image-based representations and does not rely on game-specific rules, it has not yet been validated on other domains. This includes other physics-based puzzle games, non-physics PCG domains such as platformers or roguelikes, and more complex 3D environments. Empirical evaluation across diverse game types is necessary to confirm the broader applicability and robustness of the method.

These limitations do not undermine the overall success of the framework but rather highlight areas where refinements, extensions, or adaptive strategies could substantially increase its robustness and generalizability. They also emphasize the importance of aligning technical solutions with game design principles, ensuring that automated repair serves not only structural stability but also the broader goals of structurally consistent and reliable levels.

Chapter 6

Conclusion

6.1 Summary of Work

This thesis presented a segmentation-driven repair framework for improving the stability of AI-generated physics-based game levels, with a specific application to *Angry Birds*-style structures. The framework integrates as a post-processing step in procedural content generation (PCG) pipelines, enabling automated detection and repair of structural weaknesses.

Three segmentation architectures—U-Net, SegFormer, and YOLOv8m-Seg—were trained and evaluated for gap detection, followed by a block placement repair stage designed to reinforce unstable structures. Stability improvements were quantitatively measured using Block Velocity (BV), Block Damage (BD), and Block Destruction (BDes) metrics, and further supported through human evaluation of perceived stability.

6.2 Research Questions Revisited

This thesis set out to investigate several key questions concerning the quality, stability, and repair of AI-generated game content. The following summarizes how each research question was addressed through the work presented in previous chapters.

Main Question

What are the primary problems and limitations in AI-generated game content, particularly regarding the structural stability of levels in physics-based games?

AI-generated content, while efficient and diverse, often suffers from functional flaws that lead to structural instability, especially in physics-based environments such as *Angry Birds*. The results confirmed that instability primarily arises from unsupported structures, floating elements, and gaps between blocks produced by generative models like GANs. These flaws lead to premature collapse and reduced yield of usable levels, validating the motivation for an automated repair framework.

RQ1:

Can we automatically repair unstable AI-generated game levels to improve their structural stability without extensive human intervention?

Yes. The proposed segmentation-driven repair pipeline successfully automated both the detection and correction of unstable structures. Stability metrics, such as Block Velocity (BV), Block Damage (BD), and Block Destruction (BDes) demonstrated substantial improvements after repair, with stability improvements of up to 45% under the Block Destruction metric. Manual evaluation conducted by the author indicated that repaired levels were perceptibly more stable, with minimal additional intervention required.

RQ2:

How can segmentation-based computer vision techniques be leveraged to reliably detect structural instabilities in physics-based game levels?

Semantic segmentation proved to be an effective method for identifying structural gaps and misalignments that lead to instability. By training on labeled datasets of unstable levels, the models learned to localize weak regions with high precision and recall. This approach

allowed the framework to generalize across different layouts and materials, while avoiding the limitations of rigid rule-based detection methods.

RQ3:

Which deep learning segmentation architectures (e.g., U-Net, SegFormer, YOLOv8m-Seg) are most effective for detecting destabilizing gaps and flaws in generated levels?

Among the three evaluated architectures, YOLOv8m-Seg achieved the best overall performance, balancing accuracy and computational efficiency. It demonstrated the highest F1-score, IoU, and mAP values in gap detection tasks and led to the largest post-repair stability gains. SegFormer followed closely, while U-Net provided a simpler but less robust baseline.

RQ4:

To what extent can an automated repair pipeline improve the yield of structurally stable levels from modern Procedural Content Generation systems, and can this approach generalize to other game genres and instability types?

The repair pipeline improved the proportion of stable levels by up to 45%, substantially increasing the effective yield from PCG systems. While experiments focused on *Angry Birds*-style physics, the framework’s modular design (particularly the segmentation and block-placement components) supports adaptation to other physics-based and tile-based games with appropriate retraining and calibration. Future extensions could apply the same principles to 3D environments and multi-material interactions.

6.3 Key Contributions

The main contributions of this thesis are:

1. Development of a modular framework for repairing AI-generated levels that can integrate with existing PCG workflows.
2. Comprehensive comparison of three segmentation architectures for structural gap detection in game levels.
3. Empirical evidence that repairs can significantly increase stability while preserving structural consistency.
4. Identification of YOLOv8m-Seg as the most effective architecture for industry-ready repair pipelines.

6.4 Implications for Industry

The proposed system addresses a key bottleneck in industry-level game content production—ensuring structural stability in AI-generated content. By automating stability assessment and repair, the framework reduces manual QA effort, accelerates production cycles, and maintains high structural reliability in generated levels. Its modular nature allows it to be adapted for different game engines and design styles, making it a versatile tool for studios adopting AI-driven level generation.

An additional advantage of the proposed approach is its impact on the efficiency of PCG pipelines. Rather than relying on repeated generation cycles to obtain stable levels, the framework focuses on repairing near-valid outputs. This shifts the pipeline from a generate-and-discard paradigm to a generate-and-repair paradigm, increasing the usable yield of generated content while reducing the computational overhead associated with repeated generation attempts.

6.5 Future Work

While the proposed segmentation-driven repair framework has demonstrated strong potential in improving the stability of AI-generated *Angry Birds* levels, several avenues remain open for extending and refining the system. Future research directions focus on enhancing generalizability, adaptability, usability, and efficiency.

- **Multi-Game Generalization:** The current framework is tailored to the physics, block types, and material properties of *Angry Birds*-style environments. Extending it to other physics-based games would significantly broaden its applicability. This requires recalibrating stability thresholds, retraining segmentation models on domain-specific level structures, and adapting material choices to reflect different game engines (e.g., block weights in puzzle games or destructibility in sandbox environments). Such generalization would demonstrate the framework’s versatility across diverse genres of game design.
- **Adaptive Repair Strategies:** Our present repair logic applies uniform gap-filling policies. In practice, some levels may benefit from lighter reinforcement while others require more substantial intervention. Adaptive repair strategies could dynamically adjust based on the severity, size, and location of gaps. For example, reinforcement near a critical load-bearing joint might be stronger than reinforcement in peripheral areas. Machine learning models trained to predict the “minimum effective repair” could optimize stability while preserving original structural characteristics.
- **Interactive Designer Integration:** Although our system was designed to operate autonomously, industry adoption would benefit from a hybrid workflow where human designers remain in the loop. A graphical user interface (GUI) could display model-proposed repairs, enabling designers to accept, reject, or modify suggestions in real time. This not only preserves creative control but also builds designer trust in AI tools by making the decision-making process transparent and editable.

- **Hybrid Stability Evaluation:** The current evaluation pipeline relies primarily on in-game physics simulations, which, while accurate, are computationally expensive. A promising direction is to combine simulation-based metrics with learning-based stability predictors. For instance, lightweight neural networks could rapidly approximate stability outcomes, reserving full simulations for borderline or ambiguous cases. This hybrid approach would dramatically reduce evaluation costs and allow for scalable repair of large content libraries.
- **Automated Difficulty Preservation:** Analysis of repair outcomes showed that some repairs, especially those generated by high-recall models like U-Net, reduced the structural variability of levels. To mitigate this, future systems should explicitly model the structural impact of repairs and incorporate difficulty preservation as an optimization criterion. Difficulty-aware repair algorithms could, for example, use heuristic rules, reinforcement learning, or player performance data to ensure that stability improvements do not inadvertently oversimplify the resulting structural configurations.

In summary, future extensions of this work should aim to balance automation with adaptability and creativity, ensuring that repairs remain not only structurally effective but also consistent with both design intent and player experience.

6.6 Closing Remarks

The results demonstrate that AI-generated levels can be systematically improved to meet industry standards through automated repair processes. By bridging the gap between procedural generation and structurally stable and reliable levels, this work contributes to the broader adoption of AI in game design and offers a foundation for future research in automated content refinement.

Bibliography

- [1] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural content generation via machine learning (pcgml). *IEEE Transactions on Games*, 10(3):257–270, 2018.
- [2] Noor Shaker, Julian Togelius, and Mark J. Nelson. Procedural content generation in games. In *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016.
- [3] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1–22, 2013.
- [4] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.
- [5] Frederic Abraham and Matthew Stephenson. Utilizing generative adversarial networks for stable structure generation in angry birds. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, volume 19, pages 2–12, 2023.
- [6] Aakash Kumaran and Matthew Stephenson. An end-to-end framework for procedural level generation and validation in physics-based puzzle games. In *Proceedings of*

- the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, volume 19, pages 13–23, 2023.
- [7] Forbes Magazine. Council post: The gaming industry: A behemoth with unprecedented global reach. <https://www.forbes.com/sites/forbesagencycouncil/2023/11/17/the-gaming-industry-a-behemoth-with-unprecedented-global-reach>, 2023. Accessed: 2024-07-03.
- [8] Juego Studio. how long does it take to develop video game. <https://www.juegostudio.com/blog/how-long-does-it-take-to-develop-video-game>, 2023. Accessed: 2024-07-03.
- [9] Alba Amato. Procedural content generation in the game industry. *Game Dynamics: Best Practices in Procedural and Dynamic Game Content Generation*, pages 15–25, 2017.
- [10] A.I.Design. Rogue, 1980.
- [11] Shinjin Kang, Yoonchan Ok, Hwanhee Kim, and Teasung Hahn. Image-to-image translation method for game-character face generation. In *2020 IEEE Conference on Games (CoG)*, pages 628–631. IEEE, 2020.
- [12] Martina Mittermueller, Zhanxiang Ye, and Helmut Hlavacs. Est-gan: Enhancing style transfer gans with intermediate game render passes. In *2022 IEEE Conference on Games (CoG)*, pages 25–32. IEEE, 2022.
- [13] Matthew Kreitzer, Daniel Ashlock, and Rajesh Pereira. Automatic generation of diverse cavern maps with morphing cellular automata. In *2019 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2019.
- [14] Vikram Kumaran, Bradford Mott, and James Lester. Generating game levels for multiple distinct games with a common latent space. In *Proceedings of the AAAI*

- Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, pages 102–108, 2019.
- [15] Tiago Machado, Daniel Gopstein, Angela Wang, Oded Nov, Andrew Nealen, and Julian Togelius. Evaluation of a recommender system for assisting novice game designers. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, pages 167–173, 2019.
- [16] Munir Makhmutov. Adaptive game soundtrack generation based on music transcription. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, pages 216–218, 2019.
- [17] Lei Huang and Xing Sun. Create ice cream: Real-time creative element synthesis framework based on gpt3. 0. In *2023 IEEE Conference on Games (CoG)*, pages 1–4. IEEE, 2023.
- [18] Frederic Abraham and Matthew Stephenson. Utilizing generative adversarial networks for stable structure generation in angry birds. In *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment*, volume 19, pages 2–12, 2023.
- [19] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume 3, pages 172–186. IEEE, 2011.
- [20] Mark Hendriks, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1–22, 2013.
- [21] Flávio Coutinho and Luiz Chaimowicz. On the challenges of generating pixel art

- character sprites using gans. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 18, pages 87–94, 2022.
- [22] Vincent L Prins, Jelmer Prins, Mike Preuss, and Marcello A Gómez-Maureira. Story-world: Procedural quest generation rooted in variety & believability. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*, pages 1–4, 2023.
- [23] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N Yannakakis, and Julian Togelius. Deep learning for procedural content generation. *Neural Computing and Applications*, 33(1):19–37, 2021.
- [24] Marco Ballabio and Daniele Loiacono. Heuristics for placing the spawn points in multiplayer first person shooters. In *2019 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2019.
- [25] Lucas Ferreira and Claudio Toledo. A search-based approach for generating angry birds levels. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.
- [26] Matthew Stephenson, Jochen Renz, Xiaoyu Ge, Lucas Ferreira, Julian Togelius, and Peng Zhang. The 2017 aibirds level generation competition. *IEEE Transactions on Games*, 11(3):275–284, 2018.
- [27] Matthew Stephenson and Jochen Renz. Generating varied, stable and solvable levels for angry birds style physics games. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 288–295. IEEE, 2017.
- [28] Matthew Stephenson and Jochen Renz. Procedural generation of levels for angry birds style physics games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 12, pages 225–231, 2016.

- [29] Matthew Graves. Procedural content generation of angry birds levels using monte carlo tree search. Master’s thesis, The University of Texas at Austin, 2016.
- [30] Takumi Tanabe, Kazuto Fukuchi, Jun Sakuma, and Youhei Akimoto. Level generation for angry birds with sequential vae and latent variable evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1052–1060, 2021.
- [31] Yuxuan Jiang, Tomohiro Harada, and Ruck Thawonmas. Procedural generation of angry birds fun levels using pattern-struct and preset-model. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 154–161. IEEE, 2017.
- [32] Chathura Gamage, Vimukthini Pinto, Jochen Renz, and Matthew Stephenson. Deceptive level generation for angry birds. In *2021 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2021.
- [33] Febri Abdullah, Pujana Paliyawan, Ruck Thawonmas, Tomohiro Harada, and Fitra A Bachtiar. An angry birds level generator with rube goldberg machine mechanisms. In *2019 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2019.
- [34] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation. In *Applications of Evolutionary Computation: EvoApplications 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Istanbul, Turkey, April 7-9, 2010, Proceedings, Part I*, pages 141–150. Springer, 2010.
- [35] Hafizh Adi Prasetya and Nur Ulfa Maulidevi. Search-based procedural content generation for race tracks in video games. *International Journal on Electrical Engineering & Informatics*, 8(4), 2016.
- [36] In-Chang Baek, Tae-Gwan Ha, Tae-Hwa Park, and Kyung-Joong Kim. Toward cooperative level generation in multiplayer games: A user study in overcooked! In *2022 IEEE Conference on Games (CoG)*, pages 276–283. IEEE, 2022.

- [37] Konstantinos Mitsis, Eleftherios Kalafatis, Konstantia Zarkogianni, George Mourkousis, and Konstantina S Nikita. Procedural content generation based on a genetic algorithm in a serious game for obstructive sleep apnea. In *2020 IEEE Conference on Games (CoG)*, pages 694–697. IEEE, 2020.
- [38] Olve Drageset, Mark HM Winands, Raluca D Gaina, and Diego Perez-Liebana. Optimising level generators for general video game ai. In *2019 IEEE conference on games (CoG)*, pages 1–8. IEEE, 2019.
- [39] Adi Botea and Vadim Bulitko. Generating and solving champion-level romanian crosswords puzzles. In *2023 IEEE Conference on Games (CoG)*, pages 1–4. IEEE, 2023.
- [40] Asiih Song and Jim Whitehead. Townsim: Agent-based city evolution for naturalistic road network generation. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, pages 1–9, 2019.
- [41] Marcus Gerhold and Kristian Tijben. Computer aided content generation—a gloomhaven case study. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*, pages 1–10, 2023.
- [42] Youngrok Song, Hyunju Kim, Taewoo Yoo, Byung-chull Bae, and Yun-Gyung Cheong. An intelligent storytelling system for narrative conflict generation and resolution. In *2020 IEEE Conference on Games (CoG)*, pages 192–197. IEEE, 2020.
- [43] Cory Siler. Open-world narrative generation to answer players’ questions. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 18, pages 307–310, 2022.
- [44] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1033–1038. IEEE, 1999.

- [45] Maxim Gumin. Wavefunctioncollapse: Bitmap and tilemap generation from a single example with the help of ideas from quantum mechanics. <https://github.com/mxgmn/WaveFunctionCollapse>, 2021. Accessed: 2024-07-03.
- [46] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.
- [47] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.
- [48] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. iee, 1995.
- [49] Joao António Duro and José Valente de Oliveira. Particle swarm optimization applied to the chess game. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3702–3709. IEEE, 2008.
- [50] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural content generation via machine learning (pcgml). *IEEE Transactions on Games*, 10(3):257–270, 2018.
- [51] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- [52] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [53] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley,

- Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.
- [54] Sebastian Risi and Julian Togelius. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence*, 2(8):428–436, 2020.
- [55] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.
- [56] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine learning*, 32:41–62, 1998.
- [57] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. In *International conference on machine learning*, pages 1462–1471. PMLR, 2015.
- [58] Timothy Merino, Roman Negri, Dipika Rajesh, Megan Charity, and Julian Togelius. The five-dollar model: generating game maps and sprites from sentence embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 19, pages 107–115, 2023.
- [59] Debosmita Bhaumik, Julian Togelius, Georgios N Yannakakis, and Ahmed Khalifa. Lode enhancer: Level co-creation through scaling. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*, pages 1–8, 2023.
- [60] Eugene Chen, Christoph Sydora, Brad Burega, Anmol Mahajan, Abdullah Abdullah, Matthew Gallivan, and Matthew Guzdial. Image-to-level: Generation and repair. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 189–195, 2020.

- [61] Larry R Medsker, Lakhmi Jain, et al. Recurrent neural networks. *Design and Applications*, 5(64-67):2, 2001.
- [62] Adam Summerville and Michael Mateas. Super mario as a string: Platformer level generation via lstms. *arXiv preprint arXiv:1603.00930*, 2016.
- [63] Brian W. Kernighan and Rob Pike. *The practice of programming*. Addison-Wesley, 2006.
- [64] W Ching, S Zhang, and M Ng. On multi-dimensional markov chain models. *Pacific Journal of Optimization*, 3(2):235–243, 2007.
- [65] Sam Snodgrass and Santiago Ontanón. Learning to generate video game maps using markov models. *IEEE transactions on computational intelligence and AI in games*, 9(4):410–422, 2016.
- [66] Cameron Bolitho Browne. *Automatic generation and evaluation of recombination games*. PhD thesis, Queensland University of Technology, 2008.
- [67] François Pachet. Beyond the cybernetic jam fantasy: The continuator. *IEEE Computer Graphics and Applications*, 24(1):31–35, 2004.
- [68] Peter J Bentley, Sanjeev Kumar, et al. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *GECCO*, volume 99, pages 35–43, 1999.
- [69] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.
- [70] Sam Snodgrass and Anurag Sarkar. Multi-domain level generation and blending with sketches via example-driven bsp and variational autoencoders. In *Proceedings of the 15th international conference on the foundations of digital games*, pages 1–11, 2020.
- [71] Anurag Sarkar, Adam Summerville, Sam Snodgrass, Gerard Bentley, and Joseph Osborn. Exploring level blending across platformers via paths and affordances. In *Proceedings of*

- the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 280–286, 2020.
- [72] Omid Davoodi, Mehrdad Ashtiani, and Morteza Rajabi. An approach for the evaluation and correction of manually designed video game levels using deep neural networks. *The Computer Journal*, 65(3):495–515, 2022.
- [73] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [74] Nicolas A Barriga. A short introduction to procedural content generation algorithms for videogames. *International Journal on Artificial Intelligence Tools*, 28(02):1930001, 2019.
- [75] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.
- [76] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.
- [77] Sajad Mohaghegh, Mohammad Amin Ramezan Dehnavi, Golnoosh Abdollahinejad, and Matin Hashemi. Pcgpt: Procedural content generation via transformers. *arXiv preprint arXiv:2310.02405*, 2023.
- [78] James R Wootton. Procedural generation using quantum computation. In *Proceedings of the 15th International Conference on the Foundations of Digital Games*, pages 1–8, 2020.
- [79] James R Wootton. A quantum procedure for map generation. In *2020 IEEE Conference on Games (CoG)*, pages 73–80. IEEE, 2020.

- [80] Elaine B Barker, William C Barker, William E Burr, W Timothy Polk, and Miles E Smid. Sp 800-57. recommendation for key management, part 1: General (revised), 2007.
- [81] Makoto Matsumoto, Mutsuo Saito, Hiroshi Haramoto, and Takuji Nishimura. Pseudo-random number generation: Impossibility and compromise. *J. Univers. Comput. Sci.*, 12(6):672–690, 2006.
- [82] John L. Pfaltz. Web grammars and picture description. *Computer Graphics and Image Processing*, 1(2):193–220, 1972. ISSN 0146-664X. doi: [https://doi.org/10.1016/S0146-664X\(72\)80015-7](https://doi.org/10.1016/S0146-664X(72)80015-7). URL <https://www.sciencedirect.com/science/article/pii/S0146664X72800157>.
- [83] Hartmut Ehrig, Michael Pfender, and Hans Jürgen Schneider. Graph-grammars: An algebraic approach. In *14th Annual symposium on switching and automata theory (swat 1973)*, pages 167–180. IEEE, 1973.
- [84] Paul Merrell. Example-based procedural modeling using graph grammars. *ACM Transactions on Graphics (TOG)*, 42(4):1–16, 2023.
- [85] Columbia Metropolitan Magazine. speedtree-takes-hollywood. <https://shaggydev.com/2022/03/16/generative-grammars>, 2022. Accessed: 2024-07-03.
- [86] Benoit B Mandelbrot. Self-affine fractals and fractal dimension. *Physica scripta*, 32(4): 257, 1985.
- [87] Adrian Cristea and Fotis Liarokapis. Fractal nature-generating realistic terrains for games. In *2015 7th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games)*, pages 1–8. IEEE, 2015.
- [88] Jacob Schrum, Vanessa Volz, and Sebastian Risi. Cppn2gan: Combining compositional

- pattern producing networks and gans for large-scale pattern generation. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 139–147, 2020.
- [89] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam Smith, and Sebastian Risi. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 221–228. ACM, 2018.
- [90] Yuqian Sun, Zhouyi Li, Ke Fang, Chang Hee Lee, and Ali Asadipour. Language as reality: a co-creative storytelling game experience in 1001 nights using generative ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 19, pages 425–434, 2023.
- [91] Andrew Zhu, Lara Martin, Andrew Head, and Chris Callison-Burch. Calypso: Llms as dungeon master’s assistants. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 19, pages 380–390, 2023.
- [92] Vikram Kumaran, Jonathan Rowe, Bradford Mott, and James Lester. Scenecraft: Automating interactive narrative scene generation in digital games with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 19, pages 86–96, 2023.
- [93] Graham Todd, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius. Level generation through large language models. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*, pages 1–8, 2023.
- [94] Muhammad U Nasir and Julian Togelius. Practical pcg through large language models. In *2023 IEEE Conference on Games (CoG)*, pages 1–4. IEEE, 2023.
- [95] Shyam Sudhakaran, Miguel González-Duque, Matthias Freiberger, Claire Glanois, Elias Najarro, and Sebastian Risi. Mariogpt: Open-ended text2level generation through large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

- [96] Philip Bontrager, Aditi Roy, Julian Togelius, Nasir Memon, and Arun Ross. Deepmasterprints: Generating masterprints for dictionary attacks via latent variable evolution. In *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pages 1–9. IEEE, 2018.
- [97] Vikram Kumaran, Dan Carpenter, Jonathan Rowe, Bradford Mott, and James Lester. End-to-end procedural level generation in educational games with natural language instruction. In *2023 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2023.
- [98] Thomas Volden, Djordje Grbic, and Paolo Burelli. Procedurally generating rules to adapt difficulty for narrative puzzle games. In *2023 IEEE Conference on Games (CoG)*, pages 1–4. IEEE, 2023.
- [99] Tianye Shu, Ziqi Wang, Jialin Liu, and Xin Yao. A novel cnet-assisted evolutionary level repairer and its applications to super mario bros. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–10. IEEE, 2020.
- [100] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.
- [101] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241. Springer, 2015.
- [102] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.
- [103] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once:

- Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [104] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016.
- [105] Juan Terven, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine learning and knowledge extraction*, 5(4):1680–1716, 2023.
- [106] Ultralytics. YOLOv8. <https://github.com/ultralytics/ultralytics>, 2023. GitHub repository.
- [107] Lucas N. Ferreira, Luís Pereira, and Claudio Toledo. Procedural generation of complex stable structures for angry birds levels. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2014.